

Don Thomasson

Programación Avanzada del AMSTRAD

Descripción de la ROM. Rutinas y parámetros.

PROGRAMACION AVANZADA DEL AMSTRAD

Descripción de la ROM. Rutinas y parámetros

Programación avanzada del AMSTRAD

Descripción de la ROM. Rutinas y parámetros

Don Thomasson



ANAYA MULTIMEDIA

MICROINFORMATICA

Título de la obra original:
THE INS & OUTS OF THE AMSTRAD

Traducción de: Enrique Marco
Diseño de colección: Antonio Lax
Diseño de cubierta: Narcís Fernández

Edición publicada por acuerdo con
Melbourne House (Publishers) Ltd. Londres

Villafranca, 22. 28028 Madrid
Depósito legal: M. 35.592-1985
ISBN: 84-7614-044-4
Printed in Spain
Imprime: Anzos, S. A. Fuenlabrada (Madrid)

Indice

PARTE SEGUNDA

EL INTERFAZ

PARTE TERCERA

LAS SALIDAS

Introducción

La mayor parte de un iceberg permanece escondida, accesible sólo con un equipo apropiado. Lo mismo podría decirse de las facultades del AMSTRAD CPC464. En principio puede parecer otra máquina para juegos; sin embargo, es mucho más que eso. Como en un iceberg, es necesario un equipo especial para explorar las capacidades del sistema detalladamente. El objeto de este libro es proveer ese equipo.

A primera vista está claro que el CPC464 se sale un poco de lo corriente. Que lleve el cassette incorporado no lo hace único, aunque el hecho de estar provisto de monitor monocromo o color, incluyendo la alimentación, es un detalle que se agradece, además de otros que, aunque menos obvios, también son útiles.

Para quienes se limitan al BASIC, las prestaciones de la máquina son suficientes para recomendarla, pero para aquellos usuarios que desean profundizar más en los misterios del sistema, queda mucho por explorar.

Aquí vamos a ocuparnos principalmente del modo en que el CPC464 trabaja con los periféricos. Para que esto sea efectivo, primero debemos estudiar y comprender el sistema interno, ya que éste determina cómo añadir y controlar pequeños "remiendos" y modificaciones.

El libro está dividido en dos partes: las "entradas", que tratan con el sistema interno, y las "salidas", que tratan con los equipos externos.

Para aprovechar este libro es imprescindible poseer información adicional sobre cómo programar el Z80 por las continuas referencias a rutinas en lenguaje máquina. Para información a este respecto, consultar la bibliografía al final de este volumen.

Por último, hay que aclarar que el libro se basa en el sistema de cassette del CPC464, y algunos puntos pueden ser modificados para otras versiones.

PARTE PRIMERA

LAS ENTRADAS


MSTRAD
64k COLOUR PERSONAL COMPUTER
CPC464

1 Sistema solapado

La disposición general del *hardware* interno del CPC464 se muestra en la figura 1.1. Nótese que las flechas que conectan los bloques indican la dirección del flujo de datos o de la información de control. Esto es importante.

El Z80 está situado en el centro del sistema, y controla —directa o indirectamente— todas las acciones que realiza éste. Se comunica con 64K de RAM para lectura y escritura y con los 32K de ROM, divididos en dos bloques de 16K, sólo para lectura. La selección de ROM o RAM en las direcciones en que ambas coinciden se controla mediante señales mandadas por la matriz lógica de video *(Video Gate Array)*.

A través del interfaz de entrada/salida, la CPU se comunica directamente con el controlador del tubo de rayos catódicos (CRT), que trabaja con la matriz lógica de video para mantener la visualización en pantalla; con el PPI (interfaz de periféricos paralelo), y con el interfaz de la impresora. Esta es el área primaria de periféricos.

Después de ésta, en términos del sistema, se halla el área secundaria de periféricos. Esta no está en contacto directo con la CPU, pero se comunica con el área primaria de periféricos. Los elementos en esta área externa incluyen el cassette, que se comunica con el PPI; el teclado, y el *chip* de sonido. Este último contiene una pareja de puertos de datos; uno de estos puertos lo utiliza para comunicarse con la salida del teclado.

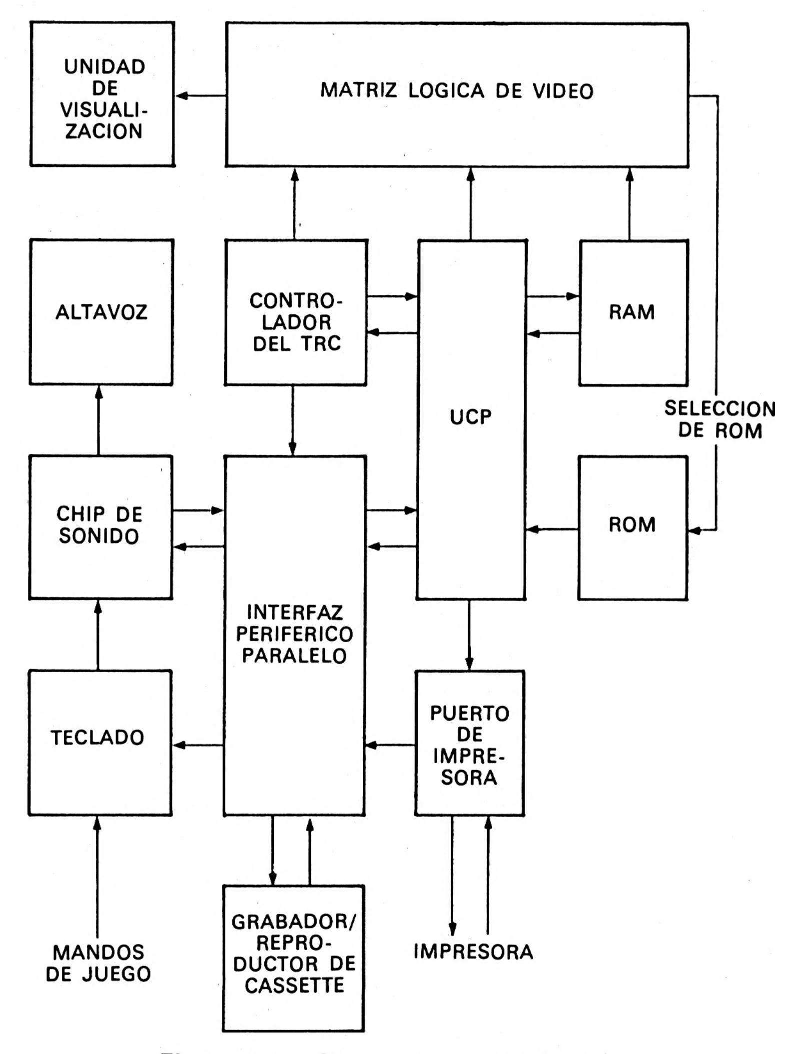


Figura 1.1. Sistema interno de *hardware.*

La CPU Z80 está controlada por un reloj de 4 MHz, pero para solapar los accesos a la memoria desde el procesador y el sistema de visualización es necesario retardar los accesos desde el procesador, reduciéndose la frecuencia efectiva del reloj a 3,3 MHz.

En términos de *hardware* el sistema de trabajo es muy económico, e incluso te preguntarás cómo puede competir con sistemas que tienen el triple de componentes. La respuesta se debe, en gran parte, al programa de control del sistema, que está organizado con más esmero de lo usual y mostrado ingeniosamente. Es cierto que un sistema mayor puede hacer cosas que el CPC464 no puede, pero

muchas de esas facultades pueden ser implementadas externamente; de este modo sólo pagas por aquellas que te gustan. Muchos usuarios podrán ver estas facilidades como adornos innecesarios, mientras buscan rasgos más allá del alcance de otras máquinas.

Los programas del sistema están divididos en nueve grupos:

a) **Controlador de teclado** *(Key Manager)*

Inspecciona el teclado, genera caracteres, implementa las expansiones de las teclas de función, comprueba el *break*, lee los *joysticks*.

b) **Unidad de visualización de texto** *(Text VDU)*

Escribe caracteres en la pantalla, maneja el cursor y responde a los códigos de control.

c) **Unidad de visualización de gráficos** *(Graphics VDU)*

Pinta y comprueba los puntos, traza líneas.

d) **Bloque de pantalla** *(Screen Pack)*

Interrelaciona las rutinas b) y c) con el *hardware* de pantalla y ejecuta las funciones comunes de ésta.

e) **Controlador de sonido** *(Sound Manager)*

Trata con la generación del sonido.

f) **Controlador de cassette** *(Cassette Manager)*

Trata con las funciones del cassette.

g) **Núcleo** *(Kernel)*

Es el corazón del sistema operativo, maneja interrupciones, programas, selección de ROM y ejecución de programas.

h) **Bloque de máquina** *(Machine Pack)*

Maneja la impresora y el control general del *hardware*.

i) **Saltador** *(Jumper)*

Controla los accesos a las rutinas mediante la tabla de saltos.

Con esto se cubre el sistema operativo, y ocupa principalmente la parte inferior de la ROM. El intérprete de BASIC, si se usa, está situado en la parte superior de la ROM, pero puede ser reemplazado por otros programas alternativos en ROM externa.

El acceso a las rutinas del sistema se hace mediante la llamada tabla de saltos *(jumpblock)*, que da acceso a varios puntos de entrada. Esta tabla quedará sin alterar en versiones futuras del sistema, pero los puntos de acceso directo pueden cambiar.

El sistema de memoria

La figura 1.2 muestra un bosquejo del mapa de memoria, pero hay muchos pequeños detalles que no pueden ser mostrados de este modo. (Nótese que todas las direcciones de memoria están en notación hexadecimal.)

El solapamiento de la ROM y RAM en la parte superior e inferior del mapa es una característica notable. Puesto que la técnica de conmutación de bancos suscitó problemas en el pasado, las implicaciones que se deriven las examinaremos más cuidadosamente.

Toda escritura de memoria se hace en la RAM. La ROM no se utiliza para escritura. Esto no es necesario explicarlo...

Si una ROM es inhibida, una memoria de lectura se realiza en esa área en la RAM. Uno o ambos de estos bloques de ROM pueden ser activados independientemente, en cuyo caso éstos suministran los datos para una lectura en la dirección de memoria correspondiente.

Las dos áreas de ROM están implantadas en un solo componente, una ROM de 32K, cuyas líneas de dirección están organizadas de la siguiente forma:

1. Si las líneas de dirección A14 y A15 quedan en bajo*, el bit de dirección A14 hacia la ROM es falso, y la ROM inferior (0000-3FFF), si está habilitada, es direccionada.
2. Si las líneas de dirección A14 y A15 son verdaderas, el bit de dirección A14 hacia la ROM es verdadero, y la ROM superior (C000-FFFF), si esta habilitada, es direccionada.
3. Si difiere el estado de las líneas de dirección A14 y A15, la ROM está inactiva.

---

* Una línea en bajo corresponde al cero lógico, y al uno lógico si está en alto. [*N. del T.*]

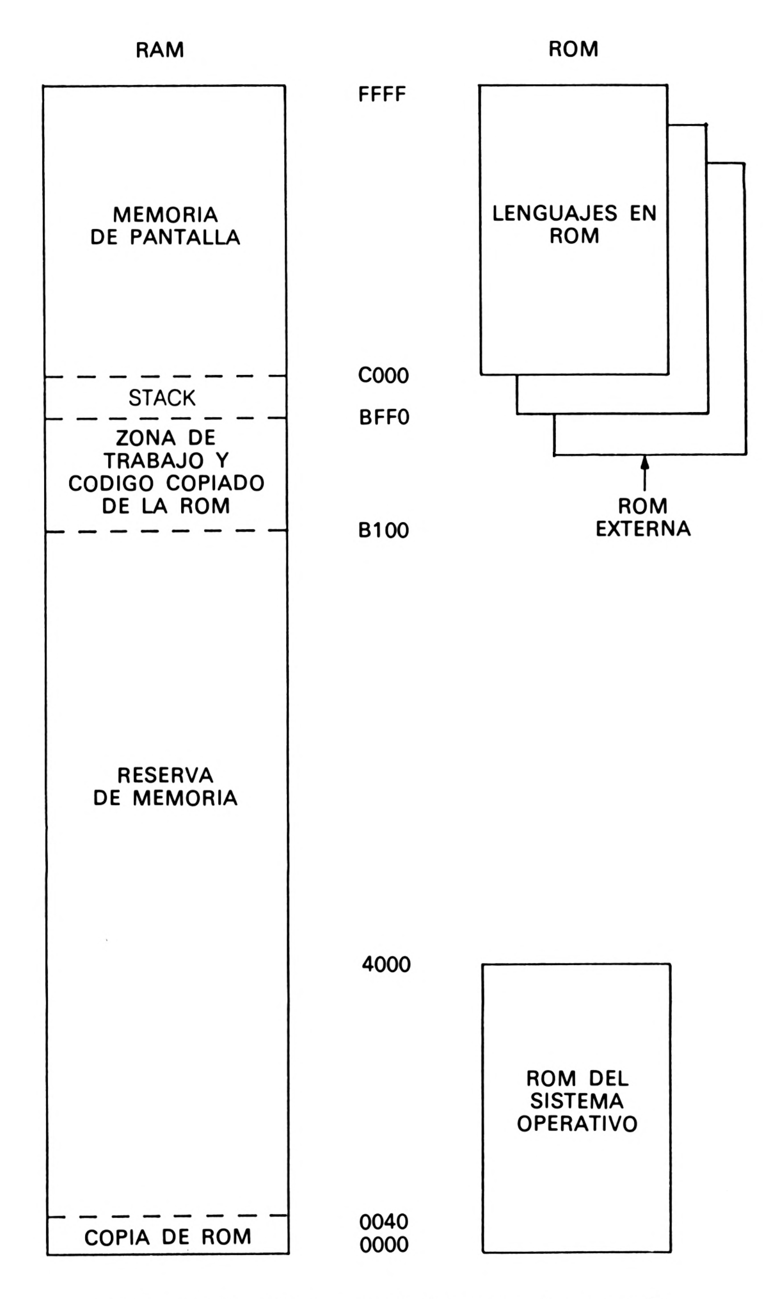


Figura 1.2. Vista general del mapa de memoria.

Las líneas de habilitación de la ROM vienen desde la matriz lógica de video, la cual deberá poder acceder a la memoria de pantalla entre los accesos a la memoria desde la CPU. Los datos importantes salen directamente hacia la matriz lógica de video, en los bits 2 (ROM inferior) y 3 (ROM superior) de una palabra de control *(control word)*. De cualquier forma se aprecia que el acceso directo es innecesario e incorrecto. Los demás bits de la palabra de control tienen una función muy importante, debiéndose mantener su estado.

La RAM está situada en 8 *chips* de 64K bits de RAM.

Volviendo a los pequeños detalles del uso de la memoria, el área 0000-003F tiene los mismos datos en ROM y en RAM; el contenido de la RAM es copiado desde la ROM durante el proceso de inicialización. Esto es necesario, ya que las instrucciones RST del Z80 acceden a esta área, y debe encontrar un código válido, tanto si la ROM está habilitada como si no. Una serie de otras rutinas pequeñas han sido situadas en la misma área, y deben ser válidas en ROM y RAM, respectivamente.

Más adelante veremos que los valores de la RAM en esta área pueden ser cambiados por el programador para propósitos particulares, pero ha de hacerse con cuidado, respetando unas reglas estrictas.

Desde 0040 hacia arriba, ROM y RAM divergen: la ROM contiene rutinas del sistema operativo, mientras que la RAM guarda datos de la zona de trabajo. Por ejemplo, los programas en BASIC se almacenan desde 0170 en adelante.

El área central de la RAM es conocida como reserva de memoria *(Memory Pool)*. Sólo la zona desde 4000 hasta C000 está siempre disponible para un acceso directo. Sería absurdo, por ejemplo, colocar la pila de máquina *(machine stack)* fuera de esta área. Normalmente es más cómodo situarla desde BFFF hacia abajo.

La parte superior del área central contiene las variables del sistema y otros programas que deben ser accesibles siempre. En el extremo, el bloque desde A400 hasta la pila puede ser ocupado si la rutina "SIMBOL AFTER 0" ha sido llamada para copiar todo el juego de caracteres en RAM. Si los modelos en RAM permanecen en el nivel estándar (desde &F0 hasta &FF), la primera posición ocupada es AB40.

Gran parte de esta área por encima de AB40 está ocupada por la tabla de saltos *(jumpblock)*, que provee entradas estándar para muchas rutinas importantes. La dirección de entrada a las rutinas puede variar, pero la tabla de saltos permanecerá en el mismo lugar. Algo a tener en cuenta: es prudente hacer uso de la tabla de saltos,

incluso cuando el punto de acceso es conocido, ya que los programas podrán funcionar con otras versiones del sistema operativo.

Desde C000 en adelante, la RAM es utilizada por la pantalla, y la ROM implementa el intérprete BASIC (aunque algunas rutinas del intérprete están en la ROM inferior). Las ROM externas pueden reemplazar a la ROM del BASIC, pudiendo implementar otros lenguajes.

Esta sucinta explicación dedicada a la memoria nos ha permitido observar los rasgos generales. Seguramente te habrá alentado para indicar un estudio más detallado. En tal caso el siguiente programa será útil para tu propósito:

```
100 CLS
110 INPUT "DIRECCION DE ENTRADA";A
120 A=A-65536*(A<0)
130 N%=A-8*INT(A/8)
140 PRINT HEX$(A,4);
150 B=PEEK(A)
160 PRINT TAB(6+3*N%);HEX$(B,2)
170 A=A+1
180 N%=N%+1
190 IF N%<8 THEN 150
200 N%=0
210 PRINT
220 GOTO 140
```

La línea 120 es necesaria, puesto que si introducimos una dirección hexadecimal por encima de 7FFF convertiría a A en negativo, y en tal caso se debe sumar 65536 para transformarlo en el valor positivo correcto. De no hacerlo así, las siguientes operaciones podrían dar errores. La variable N% sitúa la impresión de valores en la columa correcta de la pantalla, y si es necesario comienza una nueva línea.

Sin embargo, al correr el programa nos sentiremos frustrados: por todos los lados veremos montones de ceros con unos pequeños grupos de códigos. Estamos observando la RAM, porque PEEK y POKE sólo acceden a la RAM, ignorando la ROM. ¡Qué contrariedad! Necesitamos una aproximación distinta.

Buscando en el voluminoso *Manual de Firmware* descubriremos llamadas que pueden resolver nuestro problema:

B9009 Habilita la ROM superior.
B903: Deshabilita la ROM superior.

B906: Habilita la ROM inferior.
B909: Deshabilita la ROM inferior.

Para estas cuatro llamadas no hay condiciones de entrada, y el estado anterior de la ROM se devuelve en el acumulador. El contenido inicial de los *flags* es destruido, pero los demás registros no son alterados. El valor devuelto en el acumulador puede ser usado para restaurar el estado anterior de la ROM mediante:

B90C: Restaura el contenido de la ROM definido por el contenido de A. La restauración destruye el contenido de AF, pero los demás registros no son alterados.

Aunque nos dicen que los demás registros no son modificados, no es del todo cierto. El registro alternativo C′ sí cambia, porque tiene guardado el valor del último dato mandado a la matriz lógica de video. Por eso se aconseja que no se use el set de registros alternativos, dado que almacenan una importante información, que puede ser requerida en cualquier momento.

Añade al programa anterior las siguientes sentencias:

```
 90 GOSUB 400
150 GOSUB 300
300 Q=INT(A/256)
310 POKE &7019,Q
320 POKE &7018,(A-256*Q)
330 CALL &7000
340 B=PEEK(&7020)
350 RETURN
400 FOR X=&7000 TO &7012
410 READ Y
420 POKE X,Y
430 NEXT
440 RETURN
450 DATA &2A,&18,&70,&CD,&00,&B9,&F5
460 DATA &CD,&06,&B9,&7E,&32,&20,&70
470 DATA &F1,&CD,&0C,&B9,&C9
```

Con estos cambios el programa muestra el contenido de la ROM superior e inferior. Para saber el porqué debemos considerar la rutina en lenguaje máquina cargada en las líneas 400-470:

```
7000  2A  18  70  LD    HL,(7018)  ;Toma el byte de
                                     dirección.
7003  CD  06  B9  CALL  B900       ;Habilita la ROM
                                     superior.
7006  F5          PUSH  AF         ;Salva el estado de
                                     la ROM.
7007  CD  06  B9  CALL  B906       ;Habilita la ROM
                                     inferior.
700A  7E          LD    A,(HL)     ;Lee byte.
700B  32  20  70  LD    (7020),A   ;Almacena byte.
700E  F1          POP   AF         ;Recupera el estado
                                     anterior de la
                                     ROM.
700F  CD  0C  B9  CALL  B90C       ;Restaura la ROM
                                     a su estado
                                     previo.
7012  C9          RET              ;Vuelve al BASIC.
```

Las líneas 300-320 ponen el valor de A en 7018/9 (como es habitual, el byte menos significativo va primero). Se llama a la rutina en código máquina y el valor de A se coloca en HL. La rutina B900 es llamada para habilitar la ROM superior, y el valor que devuelve en el acumulador se guarda en la pila para evitar daños. B9006 habilita la ROM inferior, y el byte direccionado por HL se guarda en el acumulador, mientras que el valor del acumulador se almacena en 7020. El estado anterior de la ROM se recupera de la pila y se llama a B90C para restaurar las ROM a sus estados previos.

De vuelta al BASIC, PEEK (7020) recoge el byte, y el programa continúa al igual que el original.

Pese a la simpleza de esta rutina en máquina, los puristas pueden objetar que los parámetros podrían ser pasados mediante una extensión de la línea 300 del BASIC, pero no merece la pena perder el tiempo en ello en una rutina tan corta. El número de parámetros (en este caso uno: la variable A) podría ser mandado en el acumulador, y el registro IX apuntaría al valor del parámetro, el cual habría de ser transferido al registro HL convirtiéndolo desde notación en coma flotante en el proceso. Este método para pasar parámetros es muy útil si está bien justificado.

Soslayando esta circunstancia, ya debes tener en tu cabeza una idea bastante clara del sistema de memoria. Para una buena utilización de estos nuevos conocimientos debes saber, además, otras cosas.

Pero para un principiante es tentador experimentar. Hay espacio de sobra...

Hablando de espacio, es interesante considerar la afirmación de que 42K de memoria son utilizables por el usuario. En términos hexadecimales éstos son A800, por cuanto la afirmación no es exagerada. Casi podría ser cierto decir 43K... Aunque el tratamiento exacto del área del trabajo *(Workspace)* no está demasiado claro.

Los periféricos interiores

El mapa de direcciones entrada/salida

Dirección	Salida	Entrada
00XX a 7EXX	No usar	No usar
7FXX	Matriz lógica de video	No usar
80XX a BBXX	No usar	No usar
BCXX	Selección de registro del CRTC*	No usar
BDXX	Datos del CRTC	No usar
BEXX	No usar	Reservado (estado del CRTC)
BFXX	No usar	Datos del CRTC
C0XX a DEXX	No usar	No usar
DFXX	Selección de expansión ROM	No usar
E0XX a EEXX	No usar	No usar
EFXX	Control de la impresora *(Printer Latch)*	No usar
F0XX a F3XX	No usar	No usar
F4XX	Datos del puerto A del PPI	Datos del puerto A del PPI
F5XX	Datos del puerto B del PPI	Datos del puerto B del PPI
F6XX	Datos del puerto C del PPI	Datos del puerto C del PPI

* CRTC son las siglas de Cathode Ray Tube Controller (controlador del tubo de rayos catódicos). [*N. del T.*]

F7XX	Control del PPI	Indefinido
F8XX	*Bus* de expansión	*Bus* de expansión
F9XX	*Bus* de expansión	*Bus* de expansión
FAXX	*Bus* de expansión	*Bus* de expansión
FBXX	*Bus* de expansión	*Bus* de expansión
FCXX a FEXX	No usar	No usar
FFXX	Sin usar	Sin usar

La relación de las direcciones de entrada/salida resumidas en la tabla anterior parece aún más compleja que el mapa de memoria, pero la descalificación del *hardware* es relativamente simple.

- Si el bit de dirección A15 está en bajo, la matriz lógica de video es seleccionada.
- Si el bit de dirección A14 está en bajo, el controlador del CRT es seleccionado.
- Si el bit de dirección A13 está en bajo, el número de ROM de expansión debe estar determinado.
- Si el bit de dirección A12 está en bajo, se selecciona el control de la impresora *(Printer Latch).*
- Si el bit de dirección A11 está en bajo, se selecciona el PPI.
- Si el bit de dirección A10 está en bajo, se conecta un canal de expansión.

Sólo uno de los bits desde A10 hasta A15 puede estar en bajo para una dirección dada, lo que implica el gran número de restricciones "No usar". Es especialmente importante observar esta regla para las entradas; de otro modo podrían ocurrir daños físicos.

En el caso del controlador del CRT y el PPI, los bits A8 y A9 seleccionan una función particular del dispositivo.

Ya que todos los bits de dirección antes mencionados están en el byte de dirección más significativo, la instrucción del Z80 OUT (N),A no debe usarse. Esta toma como byte más significativo de dirección el contenido del registro A, el cual debe también contener datos. IN A,(N) sí puede usarse, si el byte más significativo requerido ha sido previamente definido en el registro A, pero no es aconsejable este método.

La forma correcta para las instrucciones de entrada/salida es OUT (C), o IN r,(C), donde *r* es un registro de 8 bits. En este caso, las direcciones de entrada/salida están definidas por el contenido del registro BC. Las instrucciones de repetición INIR, INDR, OTIR,

OTDR no deben usarse, ya que utilizan el registro B como contador y podrían dar lugar a direcciones cambiadas.

Existe una limitación adicional para usar periféricos extras, que surge del modo en que se utiliza el byte menos significativo de dirección.

- Si el bit de dirección 7 está en bajo, se selecciona el sistema de disco.
- Si el bit de dirección 6 está en bajo, se selecciona una función reservada.
- Si el bit de dirección 5 está en bajo, se selecciona un canal de comunicaciones.

Para usar otros periféricos, todos estos bits deben estar en alto, pero el byte menos significativo de &FF tiene también un significado especial, llamando a F8FF para inicializar *(reset)* todos los dispositivos de expansión. Las direcciones utilizables por el usuario son:

F8E0-F8FE
F9E0-F9FF
FAE0-FAFF
FBE0-FBFF

Estas direcciones permiten un uso libre de los bits 0, 1, 2, 3, 8 y 9, dando plenitud de alcance, suponiendo que no se utilice una decodificación simplificada. Con todos estos bits, más el bit 10 en bajo, pueden ser definidos 64 canales de entrada/salida. Si cada bit estuviera relacionado con un canal en particular, no podrían ser usados más de seis canales.

Volvemos a insistir en que un uso ilegal de las direcciones de entrada/salida puede causar daños en las operaciones de entrada, ya que más de una fuente de datos puede intentar "introducirse" en el *bus*.

La matriz lógica de video («Video Gate Array»)

La matriz lógica de video tiene tres tipos de datos en la dirección 7FXX, que se distinguen por el estado de los bits 6 y 7 del dato. Las posiciones de los bits son las siguientes:

7 6 5 4 3 2 1 0
1 0 X X X X X X

Puesta de Modo y de ROM

Control de MODO: 00 MODO 0
01 MODO 1
10 MODO 2
11 Ilegal

ROM inferior: 0 Habilitada
1 Deshabilitada

ROM superior: 0 Habilitada
1 Deshabilitada

Si vale 1 borra el divisor 52 de exploración.

Reservado. Puede ser 0.

7 6 5 4 3 2 1 0
0 0 0 X X X X X

Registro puntero de la paleta.

Número del puntero de la paleta.

7 6 5 4 3 2 1 0
0 1 0 X X X X X

Memoria de la paleta.

Número de color.

7 6 5 4 3 2 1 0
1 1 X X X X X X

Reservado

Para la puesta de modo y de ROM es esencial fijar todos los bits activos a su estado correcto, lo cual implica el mantener una copia del último byte de salida. Una copia de este byte es guardada en el registro alternativo C' del Z80. De este modo, cuando se utiliza una entrada estándar de la tabla de saltos para hacer cambios se hace referencia a esta copia automáticamente, haciendo innecesaria la referencia directa al contenido del registro.

Para fijar el color, el puntero de la paleta determina qué tinta debe utilizarse, mientras que la salida a la memoria de la paleta determina el color, la cual se utiliza después para interpretar los datos leídos de la memoria de pantalla.

La función más importante de la matriz lógica de video es obtener datos de la memoria de pantalla en base a las direcciones

suministradas por el controlador del CRT, y prepararlas para una correcta transmisión a la unidad de visualización (monitor).

La necesidad de acceder a la matriz lógica de video es limitada, puesto que el sistema se encarga de todas las acciones necesarias; pero podría haber circunstancias especiales en las que se precisase una respuesta extremadamente rápida, que únicamente puede ser realizada mediante un acceso directo.

El controlador del CRT (controlador de pantalla)

El controlador del CRT HD6845S es un componente estándar de gran complejidad, y sólo lo vamos a describir en términos relativamente superficiales. Se ocupa del control de la pantalla, de variar tamaños y características, generando las direcciones de referencia de la pantalla, los impulsos de sincronismo y un carácter de cursor sintetizado cuando así se requiera.

Las acciones de este *chip* son controladas por el contenido de unos registros, dieciocho en total, de los cuales el sistema actual sólo accede a dieciséis. Para fijar estos registros es necesario hacer un *OUT* del número del registro a la dirección BCXX y de los datos a BDXX. Los estados estándar son los siguientes:

R0 Total horizontal: 63

Determina el período de barrido horizontal en 64 períodos de carácter, los cuales vienen determinados por la frecuencia del reloj (en este caso, 8MHz), dando un ancho de un carácter por microsegundo.

R1 Horizontal visualizado: 40

Determina el número de caracteres por línea (en este caso, 40), ocupando el borrado del retorno del haz 24 períodos de carácter.

R2 Posición de sincronismo horizontal: 46

El impulso de sincronización horizontal es generado seis períodos de carácter después del final de los caracteres visualizados.

R3 Ancho del sincronismo: 142

Es un número compuesto que determina el ancho de los impulsos de sincronismo horizontal y vertical. El impulso de sincronismo vertical tiene una duración de ocho líneas de barrido, mientras que el impulso de sincronismo horizontal tiene una duración de 14 períodos de carácter ($8*16 + 14 = 142$).

R4 Total vertical: U.K. 38. U.S.A. 31

El número total de líneas de caracteres menos una son diferentes para frecuencias de barrido de 50 ó 60 Hz. Cada línea de caracteres utiliza ocho líneas de barrido, por lo que las cifras exactas deben ser 39.0625 ó 32.8125, recordando que sólo se utiliza un cuadro y no dos solapados. El ajuste necesario es hecho por R5.

R5 Ajuste del total vertical: U.K. 0. U.S.A. 6

Esto añade, en el caso de los 60 Hz, seis líneas de barrido al total vertical. El total de 50 Hz es 39, mientras que el de 60 Hz es 32.75. Este es, en la práctica, aceptablemente preciso.

R6 Vertical visualizado: 25

Se especifican 25 líneas de caracteres.

R7 Posición de sincronismo vertical: U.K. 30. U.S.A. 27

Otra vez tenemos dos valores distintos. Para 50 Hz el sincronismo vertical empieza con cinco períodos de líneas de caracteres después del área visualizada, mientras que en el caso de los 60 Hz sólo puede permitirse un retraso de dos líneas de caracteres. En los 50 Hz el sincronismo vertical termina después de 38 líneas de barrido, mientras que en el caso de los 60 Hz esta sincronización termina después de 32 líneas, y es cortada entre los períodos de líneas de caracteres números 7 y 8.

R8 Modo solapado y sesgado: 0

El cero significa que no hay solapado ni sesgado del barrido.

R9 Máximo número de líneas de barrido: 7

Determina el número de líneas de barrido menos una por carácter. Si vale siete habrá ocho líneas de barrido por carácter.

R10 Comienzo del cursor: 0

El cursor no es generado, porque:

R11 Fin del cursor: 0

El cursor sintetizado por el *chip* tiene cero líneas de altura.

R12 Dirección de comienzo (byte alto): 48

R13 Dirección de comienzo (byte bajo): 0

Determinan la dirección de comienzo del rastreo de la memoria de pantalla, como 48*256 = 12288 (3000 en hexadecimal). La matriz lógica de video modifica esta dirección.

R14 Registro del cursor (byte alto): 192

R15 Registro del cursor (byte bajo): 0

Define como dirección del cursor &C000.

No es necesario decir que estos valores están tan interrelacionados que cualquier modificación puede destruir el contenido de la pantalla. Cualquier cambio debe ser planeado cuidadosamente por anticipado, teniendo en cuenta que la matriz lógica de video modificará los resultados en algunos casos. En el modo 2 sólo es necesario un bit para definir un *pixel* y sólo se necesita un byte para definir una fila de un carácter. Son necesarios 80 bytes para definir una línea de pantalla. En el modo 1 se necesitan el doble de bytes, pero hay la mitad de caracteres; por tanto, el número total de bytes leídos por línea barrida sigue siendo 80, aunque procesados de una forma diferente. Con el modo 0 sucede otra vez lo mismo.

Las direcciones provistas por el controlador del CRT han de ser procesadas antes de ser usadas. El CRTC sólo provee 40 direcciones por línea de barrido, pero se leen 80 bytes. La conversión necesaria es bastante simple, pero debe ser recordada por cualquiera que se aventure con la pantalla. En general, estas actividades es mejor olvidarlas. El controlador del CRT puede ser visto como un dispositivo inalterable, aunque los cambios de la dirección de comienzo de la memoria de pantalla pueden tener utilidad.

Aún queda una función por mencionar: la que concierne al lápiz óptico. Utiliza un borne conectado al interfaz de expansión. Una transición de estado bajo al alto en este borne provoca que la dirección actual de la pantalla se copie en R16 (byte más significati-

vo) y R17 (byte menos significativo), de donde puede ser leída. Para interpretar correctamente esta dirección es mejor experimentar, debido a la acción de la matriz lógica de video, que modifica direcciones. El siguiente esquema de la RAM de pantalla puede ayudarnos en estos casos.

En todos los modos la memoria de pantalla ocupa 16K de RAM (generalmente desde C0000 en adelante). Llamando la rutina SCR SET BASE la pantalla puede ser trasladada al 4000 en adelante, no siendo aceptadas otras localizaciones.

La memoria de pantalla está dividida en ocho bloques de 2K. Cada Línea de pantalla utiliza 80 bytes consecutivos, pero los datos de la primera línea de barrido se obtienen del bloque 0; los de la segunda, del bloque 1, etc. De este modo el carácter situado en la esquina superior izquierda está definido por los bytes C000, C800, D000, D800, E000, E800, F000, F800 (cada uno —en el modo 2— es una línea completa del carácter), y en los modos 0 y 1 parte de la información de una línea. El modo 1 necesita 2 bytes por línea, y el modo 0, 4.

En el modo 2 los *pixels* de una línea de un carácter están definidos por los bits del byte leído de la memoria de pantalla.

En el modo 1 los *pixels*, de izquierda a derecha, están determinados por los bits 3 y 7, 2 y 6, 1 y 5, 0 y 4, y lo mismo para el byte siguiente que completa la línea del carácter.

En el modo 0 el primer *pixel* lo definen los bits 1, 5, 3 y 7; el segundo *pixel* lo definen los bits 0, 4, 2 y 6 del primer byte, y lo mismo sucede con los otros 3 bytes que faltan para completar una línea de un carácter.

Ordinariamente nada de esto debe preocuparte, ya que el programa de control se preocupa de todas las conversiones e interpretaciones necesarias, pero al menos te aclarará la extraña lista de números que obtendrías si examinases la memoria de pantalla.

Como complicación final se usa un desfase, controlable por una llamada a la rutina SCR SET OFFSET (consulta la tabla de llamadas al sistema operativo que se inserta más adelante). Cambiando el desfase por &50 (80 en decimal) desplaza la pantalla una línea arriba o abajo. Esta es un área donde está justificado experimentar, pero con cuidado. Recuerda que las reglas de cada modo son diferentes: si efectúas cambios en el desfase los resultados serán también diferentes. El modo 2 puede responder a cambios unitarios, pero en el modo 1 los cambios deben ser múltiplos de 2, y en el modo 0 han de ser múltiplos de 4.

Utilizando todo esto con la precaución debida, la pantalla ofrece

muchas posibilidades interesantes, pero un uso descuidado puede crear el caos. Examina además la lista de llamadas al sistema operativo, y utilízalas también con cautela.

El interfaz paralelo de periféricos (PPI)

El PPI *(Parallel Peripheral Interface)* es otro componente están- dar. Posee tres puertos (vías de acceso) que pueden usarse para entrada o salida. Cada puerto tiene sus propias direcciones, y el registro de control, que determina la acción del puerto, tiene una cuarta dirección.

En el CPC464 la configuración estándar es:

Puerto A: Utilizado en entrada y salida. Se comunica en dos direcciones con las líneas de datos del *chip* generador de sonido. La salida se usa para ajustar el sonido. La entrada está pendiente de la salida del teclado.

Puerto B: Sólo de entrada, con la siguiente distribución:

Bit 0: Impulso de retorno de cuadro.
Bit 1: 0 si está efectuado el enlace *(link)* LK1.
Bit 2: 0 si está efectuado el enlace LK2.
Bit 3: 0 si está efectuado el enlace LK3.
Bit 4: 0 si está efectuado el enlace LK4. (Barrido de cuadro de 60 Hz.)
Bit 5: 0 si está activo el puerto de expansión.
Bit 6: 1 si la impresora está ocupada.
Bit 7: Datos del cassette.

Puerto C: Sólo de salida, con la siguiente distribución:

Bit 0: Entrada de teclado 0.
Bit 1: Entrada de teclado 1.
Bit 2: Entrada de teclado 2.
Bit 3: Entrada de teclado 3.
Bit 4: Control del motor del cassette.
Bit 5: Datos del cassette.
Bit 6: BC1 al *chip* de sonido.
Bit 7: BDIR al *chip* de sonido.

Estas distribuciones son esenciales, pero es interesante saber que la configuración del *chip* se establece enviando una palabra de control a la dirección de entrada/salida F7XX. Esta palabra tiene dos formatos, uno para establecer el modo del *chip* y otro para poner en alto o en bajo los bits del puerto C, de poco interés en este contexto.

El formato para establecer el modo es:

7 6 5 4 3 2 1 0
1 X X X X X X X

Puerto C en bajo
Puerto B
Modo para el anterior
Puerto C en alto.
Puerto A
Modo para el anterior.

El modo 0 es el básico de entrada y salida; el modo 1 es de entrada y salida con señal de sincronización *(strobe)*, y el modo 2 es mediante un *bus* bidireccional. Si el bit de un puerto es puesto a cero, el puerto trabaja en modo de salida; si el bit es puesto a uno, el puerto trabaja en modo de entrada.

$\overline{\text{STROBE}}$	1	19	MASA
D0	2	20	MASA
D1	3	21	MASA
D2	4	22	MASA
D3	5	23	MASA
D4	6	24	MASA
D5	7	25	MASA
D6	8	26	MASA
D7	9	27	MASA
SC	10	28	MASA
BUSY	11	29	SC
SC	12	30	SC
SC	13	31	SC
MASA	14	32	SC
SC	15	33	MASA
MASA	16	34	SC
SC	17	35	SC

Nota. Se utiliza un conector de 34 vías, pero la numeración es compatible con los conectores de impresora de 36 vías, en los que los bornes 18 y 36 no se utilizan.

Figura 1.3. Conector para impresora.

El byte enviado por el sistema de control de CPC464 es &82, activando el modo 0, el puerto B en entrada y los puertos A y C en salida. (El puerto C está implementado en dos mitades cuyo modo puede ser restablecido independientemente.)

El PPI es parte esencial del *hardware* y ofrece un restringido campo de acción para cambios o experimentaciones.

El puerto de la impresora

El "Biestable Centronics" *(Centronics Latch)* direccionado por EFXX es bastante simple, demasiado simple para algunos usuarios, ya que sólo lleva 7 bits de datos más uno de sincronización *(strobe)*. Algunas impresoras pueden encontrar limitadas sus capacidades, aunque sólo por una pérdida de los gráficos de bloque del tipo asociado con los sistemas Tandy. Los códigos de control superiores, empezando por el &80, también se pierden, pero mientras esto pueda ser un estorbo en algunas máquinas debido al conflicto entre los códigos de control inferiores y los códigos de la unidad de visualización (VDU), la separación de la unidad de visualización y los datos de la impresora en vías *(streams)* diferentes hacen que la pérdida sea menos importante.

Sin embargo, nótese que el bit 7 de algunos códigos en la mitad superior del ASCII se pierde, lo cual puede ocasionar a veces resultados inesperados.

Si se intenta controlar la impresora directamente, deben seguirse las siguientes reglas:

- Los bits de datos deben estar fijados por lo menos 1 μS antes del comienzo del pulso de sincronización *(strobe)*, y deben permanecer al menos hasta 1 μS después del fin de éste.
- La duración del pulso de sincronización debe ser de 1 a 500 μS.
- El impulso de sincronización no debe ser transmitido mientras la señal de "ocupado" *(Busy)* de la impresora esté activada. El bit puede ser comprobado leyendo el bit 6 en la entrada del puerto B del PPI.

La necesidad de considerar estas reglas puede ser evitada utilizando las facilidades de que nos provee el sistema operativo:

CALL BDF1: MC WAIT PRINTER

El código del carácter a enviar debe estar en el registro A. Si el carácter fue enviado correctamente la rutina vuelve con el indicador de acarreo *(carry flag)* en alto. Si el acarreo es falso, indica que el puerto no está preparado. El contenido de los registros A y BC es destruido, y los otros registros no son alterados.

Es posible desviar esta llamada a una rutina del usuario, en cuyo caso son de interés otras llamadas afines.

CALL BD28: MC RESET PRINTER

Se restaura la dirección normal de entrada a la rutina BDF1. No hay condiciones de entrada, pero el contenido de AF, BC, DE y HL es destruido.

CALL BD2B: MC PRINT CHAR

Llama a MC WAIT PRINTER, incluso si la dirección de entrada ha sido cambiada. La organización de estas llamadas permite al usuario acceder a los códigos que van a ser enviados a la impresora y aplicar cambios condicionales.

El tiempo de espera es aproximadamente de 0,4 segundos.

Otras llamadas asociadas son:

CALL BD2E: MC BUSY PRINTER

No hay condiciones de entrada. El retorno se hace con el acarreo en alto si la impresora está ocupada (o no conectada), y en bajo si la impresora está libre. Los indicadores *(flags)* se devuelven falseados, pero los registros no se modifican.

CALL BD31: MC SEND PRINTER

Se pasa el carácter contenido en el registro A a la impresora. Al retornar el acarreo está en alto, el contenido del registro A es destruido y los demás registros no son modificados. Esta llamada sólo puede ser usada si la impresora no está ocupada.

Estos son ejemplos de las facilidades de que dispone el sistema operativo para comodidad del usuario. Estas rutinas deben ser llamadas desde código máquina, ya que se han de fijar parámetros y comprobar indicadores, pero la complejidad implicada es tan baja que merece la pena intentarlo.

Queda otro punto por explicar. El CPC464 envía los códigos retorno de carro (CR) y avance de línea (LF), pero también hace masa con la línea 14 de la impresora, lo cual permite que algunas impresoras repitan el avance de línea. Para obtener un espaciado simple en el texto es necesario desconectar la línea 14, lo cual anula el efecto de los interruptores internos tipo DIL de la impresora. Esto no sucede con la impresora AMSTRAD DMP1, que tiene un interfaz distinto.

Los periféricos exteriores

El generador de sonido programable

El generador de sonido programable AY-3-8912 es un dispositivo interesante. Tiene tres canales de sonido independientes que pueden generar una onda cuadrada o ruido rosa. El volumen puede tomar dieciséis valores, desde 0 (silencio) hasta 15 (máximo), y existe un control interno de envolvente relativamente simple. Hay también dos puertos de entrada/salida, que no utiliza el sistema del CPC464.

Describir a fondo este componente ocuparía un gran espacio, pero las siguientes notas cubrirán los aspectos especiales del sistema del CPC 464.

El PSG *(Programmable Sound Generator)* tiene dieciséis registros de control, y el método de acceder a ellos es similar al utilizado con el CRTC: primero se selecciona un registro, y después los datos pueden ser leídos o escritos en él. Los modos de transferencia son seleccionados utilizando tres líneas de control, BDIR, BC1 y BC2. En el CPC464 BC2 está permanentemente en estado alto, mientras que BC1 y BDIR son controlados, respectivamente, por los bits 6 y 7 del puerto C del PPI. Con éstos se obtienen los siguientes modos:

BDIR	BC1	
0	0	Inactivo
0	1	Lee del PSG
1	0	Escribe el PSG
1	1	Dirección del biestable (selección de registro)

Los registros pueden ser resumidos así:

R0-R5: Tomados por parejas, determinan el tono en los canales A, B y C, respectivamente. Los registros pares contienen los 8 bits menos significativos del dato del tono; los registros impares contienen los 4 bits más significativos del dato del tono, dando 12 bits de datos en total.

R6: Período del ruido. 5 bits.

R7: Control de activación. El estado alto desactiva; los bits 0-2 controlan el tono de los tres canales; los bits 3-5 controlan el ruido de los tres canales; los bits 6 y 7 controlan los dos puertos (1 para salida, 0 para entrada).

R8-R10: Amplitud del canal. 5 bits. Los valores de 0 a &F dan los niveles fijos. &1X activa el control de envolvente.

R11-R12: Período de la envolvente. Una palabra de dieciséis bits.

R13: Tipo de la envolvente. Dieciséis opciones.

R14-R14: Canales de entrada/salida.

Los datos del período del tono fijados en los tres primeros pares de registros son definidos por:

Datos del período = 125000/frecuencia requerida.
El reloj conectado al generador tiene una frecuencia de 1 MHz.
El acceso al *chip* se simplifica usando la llamada:

MC SOUND REGISTER: CALL BD34

A debe contener el número de registro.
C debe contener los datos a transmitir.
Condiciones de salida: El contenido de AF y BC es destruido.
El uso de este acceso directo es más conveniente que el control del *chip* mediante las complejas instrucciones del BASIC.
El control total de la envolvente y su control automático de tono no están disponibles, pero el del *chip* es muy versátil. La opción se escoge mediante una palabra de 4 bits como sigue:

0 a 3: Ataque fuerte, debilitándose hasta silenciarse.
4 a 7: Subida hasta el máximo y después silencio.
8: Como 0-3, pero repitiéndolo.
9: Como 0-3.
10: Comienza alto, baja, sube y repite.
11: Ataque fuerte, debilitándose hasta silenciarse; después al máximo.
12: Ataque suave, sube hasta el máximo, repite.
13: Ataque suave, sube hasta el máximo y se mantiene.
14: Al revés que el 10.
15: Como el 4-7.

El contenido de los registros R11-R12 determina las escalas de tiempo para la acción de la envolvente.

Un método cómodo para experimentar con el *chip* de sonido es utilizar una rutina en BASIC para definir los parámetros, y después llamar al bloque de código máquina que facilitamos a continuación:

```
700 C5          PUSH  BC
7001 E5         PUSH  HL        ;Salva los registros.
7002 F5         PUSH  AF
7003 21 10 70   LD    HL, 7010  ;Utiliza a HL como
                                  puntero.
7006 7E         LD    A, (HL)   ;Lee el número del
                                  registro.
7007 23         INC   HL        ;Incrementa el puntero.
7008 4E         LD    C, (HL)   ;Lee el dato.
7009 CD 34 BD   CALL  BD34      ;Llama a la rutina MC
                                  SOUND REGISTER.
700C F1         POP   AF
700D E1         POP   HL        ;Restaura los registros.
700E C1         POP   BC
700F C9         RET             ;Vuelve al BASIC.
```

El programa en BASIC debe inicializar esta rutina, como se hizo en el programa de volcado anterior, introducir el número del registro en la dirección &7010 y el valor del dato en &7011 mediante POKE. En algunos casos será necesario fijar dos registros para cubrir todos los datos, y dado que cualquiera puede tener una idea distinta del programa en BASIC, dejamos su realización como ejercicio al programador.

Los tres canales salen mezclados por el altavoz que lleva incorporado el CPC464, pero por la salida estéreo los canales A y C salen uno por cada lado y el B, por ambos.

Los tonos de onda cuadrada son característicos del clarinete, pero dos o más tonos de este tipo a veces se mezclan mal. Experimentando con filtros paso-bajo en la salida estéreo, se pueden conseguir tonos mejores.

Podrían comentarse muchas más cosas de este interesante *chip*, pero con lo dicho hay suficiente para permitir y animar a experimentar.

El teclado

El teclado es del tipo corriente de conexión en matriz, utilizando diez filas de entrada y ocho columnas de salida. Las filas son alimentadas por los bits 0-3 del canal C, y las columnas son comprobadas por el puerto del PSG y el puerto A del PPI.

El número detectado al pulsar una tecla se obtiene sumando el número de la columna al número de la fila multiplicado por ocho. Por ejemplo, si pulsamos la tecla conectada a la tercera fila y segunda columna obtenemos el número 17. Las filas y columnas empiezan a numerarse desde 0.

El teclado se explora cincuenta veces por segundo; si pulsamos una tecla el número se introduce en un *buffer*, y se hace una entrada en el "mapa de bits" *(bit map)*. La tecla se considera como pulsada hasta que se suelta durante, al menos, dos exploraciones sucesivas; esto sirve de protección contra los rebotes del contacto. De todos modos, el sistema es vulnerable a la pulsación simultánea de tres teclas de las esquinas de un rectángulo de la matriz, apareciendo la tecla de la cuarta esquina como si hubiese sido pulsada normalmente. En la práctica, esto no tiene por qué crear problemas.

Sólo después de sacar el número de la tecla del *buffer* del teclado se obtiene el código, haciendo referencia a tres tablas que definen la relación entre código y fecha en función del estado de las teclas de turno. Dicho estado se determina también por el contenido del *buffer;* así no hay riesgo de que surjan errores en la transformación debido a que el estado de las teclas haya variado desde que una fuera pulsada.

Además del estado de las mayúsculas, basculado al usar la tecla CAPS LOCK, hay también un estado *Shift Lock*, que bascula al pulsar la tecla CAPS LOCK mientras se mantiene la CRTL presio-

nada, pero no es posible una transición directa de *Shift Lock* a *Caps Lock* y viceversa, sino que debe pasarse por el estado normal.

Es posible convertir ciertos códigos en cadenas de caracteres, y el sistema permite la obtención del código o de la cadena, de acuerdo a la llamada del sistema operativo utilizada.

El conector del *joystick* está cableado directamente con la matriz del teclado, y es explorado por el mismo mecanismo.

El cassette

El cassette se relaciona con el resto del sistema mediante un interfaz de tres conectores: dos para entrada y salida de datos y uno para el control del motor. Se agradece el detalle de que el control del motor no afecte a las funciones de bobinado y rebobinado, que pueden ser utilizadas en cualquier momento.

El método de grabación usado utiliza ciclos de onda cuadrada, que emplean el doble de tiempo para los bits en alto que para los bits en bajo. La velocidad de transmisión puede ser variada entre 700 y 2.500 baudios, aunque las velocidades estándar establecidas son de 1.000 y 2.000 baudios. Durante la carga el sistema deduce la velocidad de grabación utilizada examinando el tono guía, que consiste en una larga secuencia de ciclos de estado alto.

El formato general de una grabación es:

— Una cabecera.
— *N* segmentos.
— Un bloque final.

La cabecera consiste en un espacio prerregistro, 2.048 bits en estado alto, un bit cero y un byte de sincronismo. El byte de sincronismo es el &2C para una cabecera, y el &16 para un bloque de datos.

Cada segmento está formado por 256 bytes de datos y 2 bytes CRC *(Cyclic Redundancy Check*, o comprobación cíclica redundante). El Polinomio CRC utilizado es:

X↑15 + X↑12 + X↑5 + 1 equivale a: $x^{15} + x^{12} + x^{5} + 1$

La “semilla” inicial es &FFFF. El primer byte CRC es el más significativo.

El bloque final está formado por 32 bits en estado alto.

Un fichero completo comienza con una cabecera formada por 64

bytes, que determinan el modo en que son leídos e interpretados los restantes registros. El esquema de la cabecera se muestra en la siguiente tabla:

FORMATO DEL BLOQUE DE CABECERA DEL CASSETTE

Bytes 0/15:	Nombre del fichero, completado con códigos nulos.
Byte 16:	Número de bloque.
Byte 17:	Distinto de cero si es el último bloque.
Byte 18:	Tipo de fichero:
	Bit 0: En alto para protección.
	Bits 1-3: 0: BASIC INTERNO.
	1: Binario.
	2: Copia de pantalla.
	3: ASCII.
	4-7: No especificado.
	Bits 4-7: Versión (1 para ASCII, si no 0).
Bytes 19/20:	Longitud de datos. (De grabación.)
Bytes 21/22:	Localización de datos. (Dirección de comienzo cuando se grabó.)
Byte 23:	Distinto de cero si es el primer bloque.

Campos del usuario:

Bytes 24/25:	Número de bytes en el fichero.
Bytes 26/27:	Para código máquina, dirección de entrada.
Bytes 28/63:	No especificado.

Las llamadas del sistema operativo asociadas con el cassette son un poco más comprensibles que las utilizadas en BASIC. CAS NOISY inhibe los mensajes de ayuda; CAS RETURN pone un carácter de vuelta en el *buffer* de lectura, pudiendo ser examinado y leído posteriormente; CAS OUT ABANDON permite descartar un fichero de salida, mientras CAS CHECK compara un registro en cinta con el contenido de memoria, permitiendo verificar fácilmente. Estas llamadas aparecen listadas en la tabla de llamadas al sistema operativo.

En general, las opciones normales de control del cassette dan suficientes facilidades para la mayoría de los propósitos, pero es útil saber que hay otros modos posibles de trabajo. Sin embargo, éstos son casi siempre puramente locales, y pueden producir cintas que no sean utilizables por otros CPC464, lo cual es un problema.

Otro punto importante a tener en cuenta es que el servicio normal de interrupciones queda suspendido mientras se utiliza el cassette, ya que una interrupción puede destruir la temporización precisa de las señales. El reloj de tiempo transcurrido no avanza, y otras funciones controladas por las interrupciones no son ejecutadas por la ICP.

Sin embargo, a cualquier nivel usado, el sistema de cassette tiene sus ventajas. Aunque todavía hay un misterio que no ha sido aclarado. Después de grabar o recuperar datos de tipo texto, a veces hay que esperar un buen rato para que aparezcan los mensajes de ayuda. No está claro en qué función emplea el sistema operativo este tiempo muerto.

El sistema operativo

Ya hemos examinado el *hardware* del CPC464. Pasemos ahora al *firmware*. Este no es un tema fácil. Hay más de 200 puntos de entrada, y todos están en la RAM, aunque puedes alterarlos si lo deseas y acceder con tu propio código en lugar de —o además de— las rutinas normales.

Para cada punto de entrada existe una función que realizar; hay condiciones de entrada, condiciones de salida y una serie de registros y *flags* cuyo contenido es destruido. Todo esto se indica de un modo resumido, por problemas de espacio, al final de esta sección.

Algunas de las rutinas, que no necesitan condiciones de entrada ni devuelven datos, son accesibles desde BASIC. Pero como ya hemos visto, lo normal es que haya que pasar parámetros desde el BASIC al código máquina, y viceversa.

El método estándar de pasar parámetros es eficaz, pero un poco complejo. Vamos a ver primero las distintas formas en que se almacenan los datos en BASIC:

Los números enteros son almacenados en pares de bytes en binario puro, haciéndolo primero el byte menos significativo.

Las cadenas de caracteres *(strings)* utilizan un descriptor de cadena y el cuerpo de ésta. El descriptor utiliza 3 bytes; el primero define la longitud de la cadena y los otros su dirección. Sólo necesita ser pasado como parámetro el descriptor.

Los números reales se almacenan en 5 bytes en formato de coma flotante; primero la mantisa (delante, el byte menos significativo) y, al final, el byte del exponente. Convertirlos a la forma entera, o intentar realizar cálculos con ellos, no es tarea fácil.

Afortunadamente los parámetros de un CALL se ceden de un modo simplificado. Los enteros son pasados como una palabra de 2 bytes con signo, totalmente inalterados. Un número real se convierte a la forma entera sin signo. Una variable se pasa con su dirección, y para las cadenas se debe dar la dirección del descriptor de la cadena.

Por tanto, cada parámetro se cede como una palabra de 2 bytes, y las palabras son almacenadas en un bloque al que apunta el registro IX, mientras que el número de parámetros se almacena en A. Para complicar el tema, IX apunta hacia el último parámetro.

Supongamos que quieres llamar a GRA LINE ABSOLUTE, que dibuja una línea hasta el punto definido por el contenido de DE y de HL, como las coordenadas de X e Y del punto final. No puedes llamar directamente a la dirección de entrada BBF6, porque primero han de ser determinados DE y HL. El modo en que debes hacerlo es el siguiente:

A = 2 (dos parámetros).
(IX + 2) = parámetro 1 (X).
(IX) = parámetro 2 (Y).

Necesitas una rutina que copie (IX) en HL, e (IX + 2) en DE. Esto es bastante simple:

```
7000   DD   6E   00   LD    L,(IX)
7003   DD   66   01   LD    H,(IX + 1)
7006   DD   5E   02   LD    E,(IX + 2)
7009   DD   56   03   LD    D,(IX + 3)
```

Ahora ya puedes llamar a BBF6, seguido de un código de retorno (C9).

Observa que la cuenta de parámetros en A no se utiliza. Asumimos que es correcta.

Quizá ahora entiendas por qué consideramos fácil pasar parámetros en las anteriores rutinas BASIC/código máquina mediante *pokeado* en las direcciones de memoria adecuadas. Tal vez no sea muy decoroso desde el punto de vista del BASIC, pero generalmente es más simple. Es posible crear una rutina general para introducir los parámetros, en cuyo caso la eventual llamada podría ser uno de los parámetros, aunque normalmente es más cómodo utilizar POKE y PEEK.

No es tarea fácil utilizar las rutinas del sistema operativo sólo desde el BASIC. Para llegar más lejos con el CPC464 debes utilizar el código máquina.

Como verás, hay muy pocas entradas al sistema operativo que puedan ser usadas sin problemas desde el BASIC. Sin embargo, muchas de ellas son accesibles mediante instrucciones en BASIC. Examina a fondo la lista y podrás extraer las que te sean útiles.

Una última advertencia sobre este tema: el BASIC parece capaz de restablecerse sin dificultades después de un CALL, pero en muchos casos es útil salvar los registros importantes en la pila antes de llamar a una rutina del sistema operativo e inicializarlos después, por si acaso...

El área RST

Ya han sido mencionadas las rutinas contenidas en el área RST, y necesitarás saber algo sobre ellas si quieres sacar partido de las ventajas que ofrecen. Las entradas útiles son:

0000 Entrada de arranque (también es accesible por el código de operación &C7). El uso de esta entrada provoca una reinicialización completa.

0008 Accesible mediante &CF, que ha sido convertido en un pseudocódigo de operación. Los 2 bytes que siguen a &CF son tomados como datos, utilizando la norma estándar de hacerlo primero con el byte menos significativo. Los bits de 0 a 13 definen una dirección de la cuarta parte inferior de la memoria. El bit 14 controla la ROM inferior (poniéndolo a 1 la deshabilita), y con el bit 15 se controla del mismo modo la ROM superior. Por tanto, es posible conmutar bancos y saltar en respuesta a una única llamada, semejando acciones simultáneas. Por ejemplo, CF F2 87 saltará a 07F2 con la ROM inferior habilitada y la ROM superior deshabilitada.

000B Igual que 0008, pero los 2 bytes de datos están en HL.

000E Salta a una dirección definida en BC.

0010 Accesible mediante &D7. Esta llamada es conocida como SIDE CALL. Los bytes de definición van detrás de &D7. Los bits 0-12 se suman a C000 para definir una dirección de la ROM superior, mientras que los bits 13-15 definen una ROM alternativa en términos de un desplazamiento desde la dirección de selección de la ROM principal. Esto permite acceder directamente a un punto de una ROM alternativa dada.

0013 Conocida como SIDE PCHL. Es como 0010, pero los 2 bytes de definición van en HL.

0016 Conocida como PCDE. Salta a una dirección definida en DE.

0018 Accesible por &DF. Conocida como FAR CALL. Los 2 bytes que siguen a &DF indican la dirección de una definición de 3 bytes, en los que los dos primeros bytes indican la dirección de memoria requerida. El tercer byte (YY) selecciona una ROM alternativa según las siguientes reglas:

YY = 0 a &FB:	Selecciona la ROM YY: habilita la superior y deshabilita la inferior.
YY = &FC:	No cambia la ROM: habilita la superior y la inferior.
YY = &FD:	No cambia la ROM: habilita la superior y deshabilita la inferior.
YY = &FE:	No cambia la ROM: deshabilita la superior y habilita la inferior.
YY = &FF:	No cambia la ROM: deshabilita la superior y la inferior.

001B FAR CPHL. Como 0018 pero la dirección está en HL, YY está en C.

001E CPHL. Salta a la dirección definida en HL.

0020 RAM LAM. Equivale a LD A,(HL), pero siempre accede a la RAM, cualquiera que sea el estado de la ROM. Accesible mediante &E7.

0023 FAR ICALL. HL contiene la dirección de los 3 bytes de definición, de los que los dos primeros definen una dirección y el tercero contiene YY, como se indicó antes.

0028 FIRM JUMP. Accesible mediante &EF. Los 2 bytes que siguen a &EF definen una dirección de la ROM inferior o de la RAM central. Observa que esto es un salto, no una llamada, y se debe proporcionar una dirección de retorno en la parte superior de la pila *(stack)*.

Todas estas rutinas son copiadas desde la ROM a la RAM durante la inicialización, y no deben ser alteradas. Ahora vamos a llegar a un punto donde sí es posible hacer un «remiendo».

0030 Si se entra en este punto (mediante &F7) con la ROM inferior habilitada, se ejecutan las siguientes acciones:

— Se deshabilitan las interrupciones.
— Se copia el estado actual de la ROM en 002B.
— Se deshabilita la ROM inferior.
— Se habilitan las interrupciones.
— Se salta a 0030 (en RAM).

Por tanto, es posible situar una rutina del usuario en el área 0030-0037 con la certeza de que será ejecutado si se entra por 0030, tanto si la ROM inferior está habilitada como si no. La posición 0030 en RAM se inicializa con &C7 para proteger los casos en que no se ha escrito una rutina. El sistema operativo estándar no llama a 0030.

El Z80 se pone para operar en el modo de interrupción 1, así responde a una señal de interrupción llamando a la entrada 0038. Ya que tanto ROM como RAM pueden estar activadas cuando ocurra la interrupción, ambas memorias deben guardar un salto al manipulador de interrupciones. Si el manipulador no puede reconocer la fuente de la interrupción, llama a 003B, que normalmente guarda un código de operación de retorno en ROM y RAM. La llamada se hace con la ROM deshabilitada; de este modo el usuario puede introducir su propio código en RAM entre 003B y 003F. El código será normalmente un salto a una rutina de tratamiento almacenada en cualquier parte.

La entrada a 0038 podría —pero no debe hacerse— llamarse mediante el código de operación &FF.

Para hacer un uso intensivo de estas prestaciones estamos obligados a reflexionar cuidadosamente y familiarizarnos con el juego de instrucciones del Z80. Sin embargo, las peores consecuencias de los errores pueden ser subsanadas mediante una reinicialización del sistema.

Las entradas a la tabla de saltos ("jumpblock")

Al final de esta sección se facilita una lista con más de doscientas llamadas al sistema operativo. Generalmente son accesibles a través de las entradas del *jumpblock*, que están almacenadas en RAM, y por tanto pueden ser modificadas. Estas nos proveen de un utensilio de programación muy potente.

Por ejemplo, supón que quieres alterar la rutina que manda códigos a la impresora. A la entrada de MC PRINT CHAR se accede desde BD2B. En BD2B hay el siguiente código:

CF F2 07

Como habrás observado, origina un salto a 07F2 de la ROM inferior, donde se almacena la rutina que controla la impresora. De este modo, el enlace puede ser alterado para que realice un salto a una rutina completamente diferente definida por el usuario, que se ejecutará en lugar de la rutina estándar. Los detalles de este procedimiento se dan en otra parte del libro, en relación con un sistema alternativo de *hardware* que forman un interfaz de impresora de 8 bits.

Puesto que el código modificado es almacenado en RAM, y preferiblemente en la mitad central del rango de direcciones, la instrucción CF no es necesaria, y un simple código de salto &C3 es suficiente.

Es evidente que algunas llamadas al sistema operativo no deben ser alteradas, ya que las consecuencias del cambio podrían ser muy amplias. Los experimentos deben hacerse siempre con cuidado.

Interrupciones y sucesos ("events")

El CPC464 utiliza cuatro interrupciones temporales estándar:

- El marcador rápido *(Fast Ticker)* sucede trescientas veces por segundo.
- El temporizador de sonido *(Sound Timer)* sucede cien veces por segundo.
- El retorno de cuadro *(Frame Flyback)* sucede con la frecuencia del barrido vertical.
- El marcador *(Ticker)* sucede cincuenta veces por segundo.

Estas son las interrupciones normales del tipo enmascarable. No hay opciones para tratar las interrupciones no enmascarables (NMI), las cuales provocan un salto a 0066. Este salto puede ir a ROM o RAM, de modo que las consecuencias son impredecibles.

Los sucesos son rutinas cortas que se ejecutan por interrupciones o por autorización del programa prioritario. Los sucesos activados por interrupciones se llaman "sucesos asíncronos", y los activados

por acción del programa se llaman “sucesos síncronos”. Normalmente se utiliza una lista de espera llamadas a sucesos, con listas separadas para los dos tipos, pero los “sucesos exprés” son ejecutados tan pronto como son llamados. Estos sucesos deben ser tan cortos como sea posible, y no deben habilitar interrupciones o destruir el contenido de los registros IX e IY.

Los sucesos están definidos por los “bloques de sucesos”, que son inicializados mediante la llamada a la rutina del sistema operativo KL INIT EVENT en base a los datos suministrados. Estos datos incluyen la dirección y número de ROM de la rutina que debe ejecutarse, y la localización del bloque del suceso, que tiene una longitud de 7 bytes. También debe definirse la clase de suceso utilizando un byte definido del siguiente modo:

El bit 0 está en alto si implica una “dirección próxima”, y falso si implica una “dirección lejana”. (Véanse notas en el apartado del área RST.)

Los bits del 1 al 4 sólo son relevantes en los sucesos síncronos, en los que definen el nivel de prioridad.

El bit 5 es siempre cero.

El bit 6 se pone a uno para un suceso exprés.

El bit 7 se pone a uno para un suceso asíncrono.

KL INIT EVENT crea un bloque de sucesos de la forma:

Bytes 0, 1:	Puntero de posición del siguiente bloque.	
Byte 2:	Cuenta de sucesos pendientes	Invertidos en la documentación del CPC464
Byte 3:	Clase de suceso.	
Bytes 4, 5:	Dirección de la rutina del suceso.	
Byte 6:	Selecciona el estado de ROM requerido.	

Los bytes 7 en adelante deben ser preparados por el usuario para suministrar parámetros.

Para permitir a la rutina del suceso recogerlos, se debe suministrar la dirección del bloque del suceso.

Cuando un suceso es inicializado se incrementa la cuenta del byte 3, decrementándose cuando se ha ejecutado el suceso. Se puede deshabilitar un suceso poniendo el byte de cuenta a −64. Si la cuenta vale cero, no se decrementa, y si vale &7F, no se incrementa.

Hay disponibles llamadas al sistema operativo para armar y

desarmar sucesos. Los programas prioritarios deben comprobar a intervalos regulares los posibles sucesos síncronos pendientes.

Para los sucesos asíncronos, el bloque de sucesos se etiqueta con un bloque de cabecera apropiado según el tipo de interrupción involucrada. Para interrupciones de marcador *(Ticker interrupts):*

Bytes 0, 1:	Encadenamiento de marcas (puntero de enlace).
Bytes 2, 3:	Cuenta de las marcas.
Bytes 4, 5:	Valor inicial de la cuenta (valor de recarga).
Bytes 6 en adelante:	Bloque del suceso.

La cuenta toma el valor inicial de los bytes 4 y 5 cuando se inicia un suceso, y entonces es decrementada por cada aparición de la interrupción de marcador. Cuando la cuenta alcance el valor cero, el suceso es otra vez inicializado, y la cuenta toma el valor de recarga. De este modo si el valor de "recarga" es 5, el suceso será inicializado cada cinco interrupciones de marcador, o diez veces por segundo.

Para las interrupciones *Frame Flyback* y *Fast Ticker,* sólo se añade un encadenador (puntero de enlace) al bloque del suceso.

Observa que los bloques de sucesos y de interrupciones son preparados automáticamente por las llamadas del sistema operativo, y no deben ser modificados por el usuario excepto cuando se indique lo contrario.

En un primer contacto, el sistema de sucesos y de interrupciones desanima por su complejidad, pero su potencial es enorme. Una aplicación de estas facultades es, por ejemplo, mostrar la hora del día en una esquina de la pantalla, mientras se ejecuta otro programa. Un usuario inspirado puede llenar por completo el área de la RAM intermedia con bloques de sucesos y sus rutinas; pero esto puede conducir a una situación en la que se exija al ordenador realizar lo imposible. Sólo hay un procesador, y sólo puede ejecutar un trabajo determinado en un tiempo dado. Cada suceso lo distrae de su tarea prioritaria, consumiendo tiempo de ejecución y reduciendo casi a cero el tiempo de proceso de la tarea principal.

En particular, el *Fast Ticker* ha de utilizarse con precaución. Si el suceso que llame, consume, por ejemplo, 330 microsegundos de ejecución, absorberá un décimo de tiempo de proceso, reduciendo en la misma proporción la velocidad de ejecución del proceso preferente.

Todos los bloques de interrupciones y de sucesos deben estar en la mitad central de la RAM, de forma que sean siempre accesibles. Es posible definir un "área segura" desde el BASIC mediante la instruc-

ción MEMORY, que puede utilizarse para crear un área de almacenamiento protegida. Más tarde examinaremos esto detalladamente.

Las llamadas al sistema operativo

Las llamadas al sistema operativo identificadas por AMSOFT se han listado aquí de la manera más concisa posible. Los detalles adicionales se pueden encontrar en el libro AMSTRAD CPC464 *Firmware*. (Véase bibliografía.)

Cada llamada se identifica por un nombre y una dirección de entrada de la tabla de saltos. Cada entrada de esta tabla define la dirección de la rutina actual, y estas direcciones pueden usarse cuando es importante la velocidad, pero la ROM debe ponerse en su estado correcto, aunque las entradas de la tabla de saltos fijan automáticamente el estado requerido.

Hay que mencionar que AMSTRAD garantiza las entradas de la tabla de saltos, pero no las entradas directas, que pueden ser cambiadas en versiones posteriores de la ROM.

Hay una sección de la tabla de saltos de la cual AMSOFT no describe los nombres de las llamadas o las funciones. Investigando algunas de esas entradas vemos que hay un convertidor de variables enteras a coma flotante y otras funciones muy útiles. Estas son utilizadas por el BASIC, pero no es factible una explicación más amplia de sus características, ya que requieren un conocimiento interno del intérprete BASIC.

Puede asumirse que los registros no mencionados en las condiciones de salida son preservados. No obstante, debe tenerse en cuenta que esto no incluye a los registros alternativos, que no son utilizables por el usuario.

Llamadas al sistema operativo

KM INITIALISE: BB00

Inicializa completamente el director de teclado; todas las variables, *buffers* e indirecciones especiales se pierden.

No hay condiciones de entrada.

Condiciones de salida: el contenido de AF, BC, DE y HL es destruido.

KM RESET: BB03

Reinicializa en las indirecciones director de teclado y *buffer*.

No hay condiciones de entrada.

Condiciones de salida: el contenido de AF, BC, DE y HL es destruido.

KM WAIT CHAR: BB06

Espera un carácter del *buffer* del teclado o de una cadena de expansión.

No hay condiciones de entrada.

Condiciones de salida: el código del carácter es devuelto en A y el acarreo queda en alto. El estado de los demás indicadores es alterado.

KM READ CHAR: BB09

Toma un carácter del *buffer* del teclado o de una cadena de expansión si está disponible; si no, no espera.

No hay condiciones de entrada.

Condiciones de salida: si el carácter está disponible el código se introduce en A y el acarreo queda en alto; si no, el acarreo queda en bajo y A se devuelve alterado. Los demás indicadores quedan alterados.

KM CHAR RETURN: BB0C

Reinserta un carácter en el *buffer* del teclado, para que se pueda volver a leer más tarde, permitiendo comprobar el carácter sin que se pierda.

Los caracteres han de ser devueltos de uno en uno. El carácter devuelto debe ser leído antes de devolver otro.

Condiciones de entrada: A contiene el carácter que va a ser devuelto.

Condiciones de salida: son preservados todos los registros e indicadores.

KM SET EXPAND: BB0F

Determina una cadena de expansión relacionada con un código de palabra clave *(token)*.

Condiciones de entrada: B contiene la palabra clave de expansión que va a usarse, y C contiene la longitud de la cadena.

Condiciones de salida: el acarreo queda en alto si se determina la palabra clave; si no, queda en bajo.

Nota: La serie debe estar en RAM, no en ROM.

KM GET EXPAND: BB12

Lee un carácter de una cadena de expansión. El primer carácter es el número 0.

Condiciones de entrada: A contiene la cadena de expansión y L el número del carácter.

Condiciones de salida: si lo ha leído bien, el acarreo queda en alto y el código del carácter se devuelve en A; si no, el acarreo queda en bajo y A es alterado. DE y los demás indicadores quedan alterados.

KM EXP BUFFER: BB15

Se asigna un *buffer* para cadenas de expansión y se inicializa con la cadena de expansión por omisión.

Condiciones de entrada: DE contiene la dirección del *buffer*; HL contiene su longitud.

Condiciones de salida: el acarreo queda en alto salvo si el *buffer* es demasiado corto. A, BC, DE, HL y los demás indicadores quedan alterados.

KM WAIT KEY: BB18

Espera por un código del *buffer* del teclado.

No hay condiciones de entrada.

Condiciones de salida: el acarreo queda en alto y A contiene el carácter o la palabra clave de expansión. (Las palabras claves no son expandidas.)

KM READ KEY: BB1B

Tome un código del *buffer* del teclado si hay uno disponible.

No hay condiciones de entrada.

Condiciones de salida: si estaba disponible un código el acarreo queda en alto y A contiene el código del carácter o la palabra clave de expansión; si no, el acarreo queda en bajo y A es alterado. Los demás indicadores quedan alterados.

KM TEST KEY: BB1E

Comprueba si está pulsada una tecla específica.

Condiciones de entrada: A contiene un número de tecla.

Condiciones de salida: si la tecla está pulsada el indicador de cero queda en bajo; si no, queda en alto. El acarreo queda siempre en alto, C contiene el estado actual de las teclas Shift y Control. A, HL y los demás indicadores quedan alterados.

KM GET STATE: BB21

Comprueba los estados de Caps y Shift LOCK.

No hay condiciones de entrada.

Condiciones de salida: L contiene el estado de Shift Lock; H contiene el estado de Caps Lock. Si el Lock está activado, el estado es &FF; si no, es 0. AF queda alterado.

KM GET JOYSTICK: BB24

Comprueba el estado del *joystick*.

No hay condiciones de entrada.

Condiciones de salida: A y H contienen el estado del *joystick* 0; L contiene el estado del *joystick* 1. Los indicadores quedan alterados.

Los bytes de información tienen la siguiente configuración:

Bit 0: ARRIBA.

Bit 1: ABAJO.

Bit 2: IZQUIERDA.

Bit 3: DERECHA.

Bit 4: FUEGO 2.

Bit 5: FUEGO 1.

Bit 6: Pulsador de repuesto.

Bit 7: 0.

KM SET TRANSLATE: BB27

Determina el código o palabra clave generado por una tecla dada.

Condiciones de entrada: A contiene el número de la tecla; B contiene el código o palabra clave.

Condiciones de salida: AF y HL quedan alterados.

Los siguientes valores de palabras claves tienen un uso especial:

80-9F: Palabras claves de expansión.

E0-FC: Códigos de edición del sistema.

FD: Caps Lock.

FE: Shift Lock.

FF: Ignorado. (Teclas sin significado.)

KM GET TRANSLATE: BB2A

Traduce el significado de una tecla sin Shift ni Control.

Condiciones de entrada: A contiene el número de tecla.

Condiciones de salida: A contiene la traducción; HL y los indicadores quedan alterados.

KM SET SHIFT: BB2D

Determina el significado de una tecla con Shift, y sin Control.

Condiciones de entrada: A contiene el número de tecla; B contiene el significado.

Condiciones de salida: AF y HL quedan alterados.

Nota: Véase KM SET TRANSLATE para los códigos con un significado especial.

KM GET SHIFT: BB30

Traduce el significado de una tecla con Shift y sin Control.

Condiciones de entrada: A contiene un número de tecla.

Condiciones de salida: A contiene la traducción; HL y los indicadores quedan alterados.

KM SET CONTROL: BB33

Determina el significado de una tecla con Control.

Condiciones de entrada: A contiene el número de tecla y B el significado.

Condiciones de salida: AF y HL quedan alterados.

Nota: Véase KM SET TRANSLATE para los códigos con un significado especial.

KM GET CONTROL: BB36

Traduce el significado de una tecla con Control.

Condiciones de entrada: A contiene el número de la tecla.

Condiciones de salida: A contiene el significado; HL y los indicadores quedan alterados.

KM SET REPEAT: BB39

Determina si se asigna repetición a una tecla.

Condiciones de entrada: A contiene el número de tecla; B contiene &&FF si se asigna repetición; si no, contiene 0.

Condiciones de salida: AF, BC y HL están falseados.

KM GET REPEAT: BB3C

Comprueba si una tecla tiene asignada repetición.

Condiciones de entrada: A contiene el número de una tecla.

Condiciones de salida: el indicador de acarreo queda en bajo si la tecla tiene asignada repetición; si no, queda en alto. El acarreo queda en bajo. A, HL y los otros indicadores quedan alterados.

KM SET DELAY: BB3F

Determina la demora de comienzo y el período de repetición de las teclas.

Condiciones de entrada: H contiene la demora de comienzo; L contiene el período de repetición. Los valores se dan en cincuentavos de segundo. Un valor de 0 se interpreta como 256.

Condiciones de salida: AF queda alterado.

KM GET DELAY: BB42

Comprueba la demora de comienzo y el período de repetición de las teclas.

No hay condiciones de entrada.

Condiciones de salida: H contiene la demora de comienzo; L contiene el período de repetición. Los valores se dan en cincuentavos de segundo. AF queda alterado.

KM ARM BREAKS: BB45

Habilita los sucesos del *break*.

Condiciones de entrada: DE contiene la dirección de la rutina del *break*; C contiene la dirección de ROM seleccionada para la rutina.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

KM DISARM BREAKS: BB48

Deshabilita los sucesos del *break*. (Este es el estado por omisión.)

No hay condiciones de entrada.

Condiciones de salida: AF y HL quedan alterados.

KM BREAK EVENT: BB4B

Genera un *break* si los sucesos de éste están habilitados. Deshabilita los sucesos del *break*.

No hay condiciones de entrada.

Condiciones de salida: AF y HL quedan alterados.

Una palabra clave de un suceso de *break* (&EF) se coloca en el *buffer* del teclado. Esto genera un suceso de *break* cuando es leído del *buffer*.

TXT INITIALISE: BB4E

Inicialización completa de la unidad de visualización de texto, incluyendo todas las variables e indirecciones.

No hay condiciones de entrada.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

TXT RESET: BB51

Reinicializa las indirecciones de la unidad de visualización de texto y la tabla de control.

No hay condiciones de entrada.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

TXT VDU ENABLE: BB54

Permite mostrar datos en pantalla con la vía *(stream)* actual seleccionada.

No hay condiciones de entrada.

Condiciones de salida: AF queda alterado.

TXT VDU DISABLE: BB57

Impide mostrar datos en pantalla con la vía *(stream)* actual seleccionada (incluyendo el cursor).

No hay condiciones de entrada.

Condiciones de salida: AF queda alterado.

TXT OUTPUT: BB5A

Manda un código a la unidad de visualización de texto.

Condiciones de entrada: A contiene el código a mandar.

Condiciones de salida: todos los indicadores y registros son preservados.

TXT WRITE CHAR: BB5D

Imprime un carácter en la posición del cursor de la vía *(stream)* actual. Los códigos de control expuestos no son obedecidos, pero sí impresos. La VDU debe estar habilitada.

Condiciones de entrada: A contiene el código del carácter.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

TXT RD CHAR: BB60

Lee un carácter de la pantalla en la posición del cursor de la vía *(stream)* actual.

No hay condiciones de entrada.

Condiciones de salida: si el carácter es reconocido el acarreo queda en alto y A contiene el código del carácter; si no, el acarreo queda en bajo y A contiene 0.

Los demás indicadores quedan alterados.

TXT SET GRAPHIC: BB63

Habilita o deshabilita la opción de caracteres gráficos.

Condiciones de entrada: A vale 0 para activar la opción, y distinto de 0 para desactivarla.

Condiciones de salida: AF queda alterado.

TXT WIN ENABLE: BB66

Determina una ventana para la vía *(stream)* actual.

Condiciones de entrada: D y H contienen los márgenes izquierdo y derecho; el menor valor es el margen izquierdo. E y L contienen los márgenes superior e inferior; el menor es el margen superior.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

TXT GET WINDOW: BB69

Comprueba los límites de la ventana en la vía *(stream)* actual.

No hay condiciones de entrada.

Condiciones de salida: H contiene la columna izquierda; D, la columna derecha; L, la fila superior, y E contiene la fila inferior. El acarreo queda alterado si la ventana cubre toda la pantalla.

Nota: En las últimas dos llamadas hay que tener en cuenta que las posiciones empiezan a contar desde 0, no desde 1 como en BASIC.

TXT CLEAR WINDOW: BB6C

Borra el texto de la ventana en la vía *(stream)* actual, rellenándolo con la tinta del papel en esa vía. Sitúa el cursor en la esquina superior izquierda.

No hay condiciones de entrada.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

TXT SET COLUMN: BB6F

Mueve el cursor de la vía *(stream)* actual a una columna especificada.

Condiciones de entrada: A contiene el número de columna requerido. (De 1 en adelante.)

Condiciones de salida: AF y HL quedan alterados.

TXT SET ROW: BB72

Mueve el cursor en la vía *(stream)* actual a una fila especificada.

Condiciones de entrada: A contiene el número de la fila requerido. (De 1 en adelante.)

Condiciones de salida: AF y HL quedan alterados.

TXT SET CURSOR: BB75

Mueve el cursor en la vía *(stream)* actual a una columna y a una fila especificadas.

Condiciones de entrada: H contiene la columna y L la fila. (De 1 en adelante.)

Condiciones de salida: AF y HL quedan alterados.

TXT GET CURSOR: BB78

Comprueba la posición del cursor en la vía (*stream*) actual, así como el contador de desplazamiento de pantalla *(roll count)*.

No hay condiciones de entrada.

Condiciones de salida: H contiene la columna y L la fila; A contiene el contador de desplazamiento, que es decrementado cuando la ven-

tana es desplazada hacia arriba, y es incrementado cuando la ventana es desplazada hacia abajo. La posición del cursor devuelta no es necesariamente válida.

TXT CUR ENABLE: BB7B

Habilita el cursor para la vía *(stream)*.

No hay condiciones de entrada.

Condiciones de salida: AF queda alterado.

TXT CUR DISABLE: BB7E

Deshabilita el cursor para la vía *(stream)* actual.

No hay condiciones de entrada.

Condiciones de salida: AF queda alterado.

TXT CUR ON: BB81

Permite mostrar en pantalla el cursor en la vía *(stream)* actual.

No hay condiciones de entrada.

Condiciones de salida: todos los registros e indicadores están preservados.

TXT CUR OFF: BB84

Impide mostrar en pantalla el cursor en la vía *(stream)* actual.

No hay condiciones de entrada.

Condiciones de salida: todos los registros e indicadores están preservados.

Nota: BB7B/E están destinadas para un uso general, y se imponen a BB81/4 en lo concerniente a la supresión del cursor. BB81/4 son para uso de las ROM del sistema.

TXT VALIDATE: BB87

Comprueba si el cursor está dentro de los límites de la ventana actual.

Condiciones de entrada: H contiene la columna y L la fila (de 1 en adelante) de la posición.

Condiciones de salida: si está dentro de la ventana, el acarreo queda en alto y B alterado. Si la ventana tiene que desplazarse hacia arriba, el acarreo queda en bajo y B contiene &FF. Si la ventana tiene que desplazarse hacia abajo, el acarreo queda en bajo y B contiene 0. En todos los casos, H y L contienen la columna y la fila en la que el carácter debe aparecer. A y los demás indicadores quedan alterados.

TXT PLACE CURSOR: BB8A

Muestra un cursor en la vía *(stream)* actual. Esto permite varios cursores. Es absurdo llamar a TXT PLACE CURSOR dos veces para una posición de pantalla dada, sin que intervenga TXT REMOVE CURSOR, porque seguimos teniendo el primer cursor en esa posición.

No hay condiciones de entrada.

Condiciones de salida: AF queda alterado.

TXT REMOVE CURSOR: BB8D

Retira un cursor múltiple de la pantalla.

No hay condiciones de entrada.

Condiciones de salida: AF queda alterado.

TXT SET PEN: BB90

Define la tinta de escritura de la vía *(stream)*.

Condiciones de entrada: A contiene el número de la tinta.

Condiciones de salida: AF y HL quedan alterados.

TXT GET PEN: BB93

Comprueba la tinta de escritura en la vía *(stream)* actual.

No hay condiciones de entrada.

Condiciones de salida: A contiene el número de la tinta. Los indicadores quedan alterados.

TXT SET PAPER: BB96

Define la tinta del papel para la vía *(stream)* actual.

Condiciones de entrada: A contiene el número de la tinta.

Condiciones de salida: AF y HL quedan alterados.

TXT GET PAPER: BB99

Comprueba la tinta del papel en la vía *(stream)* actual.

No hay condiciones de entrada.

Condiciones de salida: A contiene el número de la tinta. Los indicadores quedan alterados.

TXT INVERSE: BB9C

Intercambia la tinta del papel y la de escritura en la vía *(stream)* actual.

No hay condiciones de entrada.

Condiciones de salida: AF y HL quedan alterados.

TXT SET BACK: BB9F

Determina si se muestra el papel, es decir, si se escribe en modo opaco (se muestra el papel) o en modo transparente (no se muestra).

Condiciones de entrada: A contiene 0 para el modo opaco, y un valor distinto de 0 para el modo transparente.

Condiciones de salida: AF y HL quedan alterados.

TXT GET BACK: BBA2

Comprueba si se está mostrando el papel.

No hay condiciones de entrada.

Condiciones de salida: A contiene 0 si se está utilizando el modo opaco; si no, contiene un valor distinto de 0. DE, HL y los indicadores quedan alterados.

TXT GET MATRIX: BBA5

Obtiene la dirección del modelo de un carácter.

Condiciones de entrada: A contiene el código del carácter.

Condiciones de salida: HL contiene la dirección del modelo. Si está en ROM el acarreo queda en bajo; si está en RAM el acarreo queda en alto. A y los demás indicadores quedan alterados.

TXT SET MATRIX: BBA8

Define un modelo de carácter.

Condiciones de entrada: A contiene el código del carácter a definir, y HL contiene la dirección de la matriz que va a utilizarse para definir el modelo.

Condiciones de salida: el acarreo queda en alto si el carácter es definible por el usuario; si no, queda en bajo. A, BC, DE, HL y los demás indicadores quedan alterados.

TXT SET M TABLE: BBAB

Define una tabla de modelos de caracteres. (Hasta &FF.)

Condiciones de entrada: DE contiene el código del primer carácter. HL contiene la dirección de comienzo de la tabla de modelos.

Condiciones de salida: si anteriormente no había tabla definida por el usuario, el acarreo queda en bajo, y A y HL quedan alterados. Si la había, el acarreo queda en alto, y A contiene el código del primer carácter. HL contiene la dirección de la tabla antigua. BC, DE y los demás indicadores quedan alterados.

TXT GET M TABLE: BBAE

Comprueba la dirección de una tabla de caracteres definida por el usuario y el código del primer carácter de la tabla.

No hay condiciones de entrada.

Condiciones de salida: si no hay una tabla de caracteres definida por el usuario, el acarreo queda en bajo, A y HL quedan alterados; si no, el acarreo queda en alto. A contiene el código del primer carácter de la tabla y HL contiene la dirección de comienzo de la tabla. Los demás indicadores quedan alterados.

TXT GET CONTROL: BBB1

Obtiene la dirección de la tabla de códigos de control. Esta tiene 3 bytes por código. El primer byte indica el número de parámetros que requiere el código; los otros dos indican la dirección de la rutina relacionada.

No hay condiciones de entrada.

Condiciones de salida: HL contiene la dirección de la tabla de control.

TXT STR SELECT: BBB4

Selecciona la vía *(stream)*.

Condiciones de entrada: A contiene la vía *(stream)* a seleccionar.

Condiciones de salida: HL y los indicadores quedan alterados.

TXT SWAP STREAMS: BBB7

Intercambia dos vías *(streams)*.

Condiciones de entrada: C y B contienen el número de las dos vías *(streams)*.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

GRA INITIALISE: BBBA

Inicializa la unidad de visualización de gráficos. (Todas las variables e indirecciones toman los valores por omisión.)

No hay condiciones de entrada.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

GRA RESET: BBBD

Fija las indirecciones de la unidad de visualización de gráficos a los valores por omisión.

No hay condiciones de entrada.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

GRA MOVE ABSOLUTE: BBC0

Mueve el cursor de gráficos a una posición en coordenadas absolutas.

Condiciones de entrada: DE contiene la coordenada X; HL contiene la coordenada Y.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

GRA MOVE RELATIVE: BBC3

Mueve el cursor de gráficos a una posición relativa a la posición actual.

Condiciones de entrada: DE contiene la coordenada X relativa y HL contiene la coordenada Y relativa.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

GRA ASK CURSOR: BBC6

Comprueba la posición del cursor de gráficos.

No hay condiciones de entrada.

Condiciones de salida: DE contiene la coordenada X absoluta; HL contiene la coordenada Y absoluta. AF queda alterado.

GRA SET ORIGIN: BBC9

Determina el punto de origen de coordenadas para el cursor de gráficos.

Condiciones de entrada: DE contiene la coordenada X estándar y HL contiene la coordenada Y estándar.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

GRA GET ORIGIN: BBCC

Comprueba la posición del origen de coordenadas.

No hay condiciones de entrada.

Condiciones de salida: DE contiene la coordenada X estándar y HL contiene la coordenada Y estándar.

GRA WIN WIDTH: BBCF

Define los márgenes derecho e izquierdo de la ventana de gráficos.

Condiciones de entrada: DE y HL contienen las coordenadas X estándar de los márgenes. El menor de los valores determina el margen izquierdo.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

GRA WIN HEIGHT: BBD2

Define los límites superior e inferior de la ventana de gráficos.

Condiciones de entrada: DE y HL contienen las coordenadas estándar de los límites.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

GRA GET W WIDTH: BBD5

Comprueba los márgenes derecho e izquierdo de la ventana de gráficos.

No hay condiciones de entrada.

Condiciones de salida: DE contiene la coordenada X estándar del margen izquierdo; HL contiene la coordenada X estándar del margen derecho. AF queda alterado.

GRA GET W HEIGHT: BBD8

Comprueba los límites superior e inferior de la ventana de gráficos.

No hay condiciones de entrada.

Condiciones de salida: DE contiene la cordenada Y estándar del límite superior; HL contiene la coordenada Y estándar del límite inferior. AF queda alterado.

GRA CLEAR WINDOW: BBDB

Borra la ventana de gráficos rellenándola con el color de la tinta del papel de gráficos.

No hay condiciones de entrada.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

Nota: El cursor de gráficos se sitúa en el origen de coordenadas.

GRA SET PEN: BBDE

Define una tinta para el pincel de gráficos.

Condiciones de entrada: A contiene el número de la tinta.

Condiciones de salida: AF queda alterado.

GRA GET PEN: BBE1

Comprueba el color actual del pincel de gráficos.

No hay condiciones de entrada.

Condiciones de salida: A contiene el número de la tinta. Los indicadores quedan alterados.

GRA SET PAPER: BBE4

Define el color del papel de gráficos.

Condiciones de entrada: A contiene el número de la tinta.

Condiciones de salida: AF queda alterado.

GRA GET PAPER: BBE7

Comprueba el color del papel de gráficos.

No hay condiciones de entrada.

Condiciones de salida: A contiene el número de la tinta. Los indicadores quedan alterados.

GRA PLOT ABSOLUTE: BBEA

Activa un *pixel* en coordenadas absolutas.

Condiciones de entrada: DE contiene la coordenada X absoluta y HL contiene la coordenada Y absoluta.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

GRA PLOT RELATIVE: BBED

Activa un *pixel* en una posición relativa a la posición actual del cursor de gráficos.

Condiciones de entrada: DE contiene la coordenada X relativa y HL contiene la coordenada Y relativa.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

GRA TEST ABSOLUTE: BBF0

Comprueba la tinta de un *pixel* en coordenadas absolutas.

Condiciones de entrada: DE contiene la coordenada X absoluta y HL contiene la coordenada Y absoluta.

Condiciones de salida: A contiene el número de la tinta. BC, DE, HL y los indicadores quedan alterados.

GRA TEST RELATIVE: BBF3

Comprueba la tinta de un *pixel* en coordenadas relativas a la posición actual del cursor de gráficos.

Condiciones de entrada: DE contiene la coordenada X relativa y HL la coordenada Y relativa.

Condiciones de salida: A contiene el número de la tinta. BC, DE, HL y los indicadores quedan alterados.

GRA LINE ABSOLUTE: BBF6

Dibuja una línea desde la posición actual del cursor de gráficos hasta la posición indicada, en coordenadas absolutas.

Condiciones de entrada: DE contiene la coordenada X absoluta; HL contiene la coordenada Y absoluta.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

GRA LINE RELATIVE: BBF9

Dibuja una línea desde la posición actual del cursor hasta la posición indicada, en coordenadas relativas a la posición del cursor de gráficos.

Condiciones de entrada: DE contiene la coordenada X relativa; HL contiene la coordenada Y relativa.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

GRA WRITE CHAR: BBFC

Escribe un carácter en la posición actual del cursor de gráficos.

Condiciones de entrada: A contiene el código del carácter.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

SCR INITIALISE: BBFF

Inicializa completamente el paquete de pantalla *(Screen Pack)*. Se reinicializan todas las variables y direcciones de entrada, así como el modo de pantalla y las tintas.

No hay condiciones de entrada.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

SCR RESET: BC02

Reinicializa los colores y las indirecciones del paquete de pantalla, así como la velocidad de parpadeo y el modo de escritura.

No hay condiciones de entrada.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

SCR SET OFFSET: BC05

Define un desplazamiento de pantalla.

Condiciones de entrada: HL contiene el desplazamiento requerido.

Condiciones de salida: AF y HL quedan alterados.

SCR SET BASE: BC08

Define la dirección base de pantalla. (C000 ó 4000.)

Condiciones de entrada: A contiene el byte más significativo de la dirección base.

Condiciones de salida: AF y HL quedan alterados.

SCR GET LOCATION: BC0B

Comprueba la dirección base de pantalla y el desplazamiento.

No hay condiciones de entrada.

Condiciones de salida: A contiene el byte más significativo de la dirección base; HL contiene el desplazamiento. Los indicadores quedan alterados.

SCR SET MODE: BC0E

Define el modo de pantalla.

Condiciones de entrada: A contiene el modo requerido.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

SCR GET MODE: BC11

Comprueba el modo actual de pantalla.

No hay condiciones de entrada.

Condiciones de salida: para el modo 0, el acarreo queda en alto, el indicador de 0 queda en bajo y A contiene 0. Para el modo 1, el acarreo queda en bajo, el indicador de 0 queda en alto y A contiene 1. Para el modo 2, los indicadores de acarreo y 0 quedan en bajo y A contiene 2. Los demás indicadores quedan alterados.

SCR CLEAR: BC14

Borra la pantalla con la tinta 0.

No hay condiciones de entrada.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

SCR CHAR LIMITS: BC17

Comprueba el tamaño de la pantalla en caracteres.

No hay condiciones de entrada.

Condiciones de salida: B contiene la última columna de pantalla; C contiene la última fila de pantalla. AF queda alterado.

SCR CHAR POSITION: BC1A

Calcula la dirección de pantalla de la esquina superior izquierda de la posición de un carácter.

Condiciones de entrada: H contiene la columna y L la fila, en valores físicos (de 0 en adelante).

Condiciones de salida: HL contiene la dirección de pantalla; B contiene los bytes por carácter. AF está falseado.

SCR DOT POSITION: BC1D

Calcula la dirección de pantalla de un *pixel.*

Condiciones de entrada: DE contiene la coordenada X; HL contiene la coordenada Y.

Condiciones de salida: HL contiene la dirección de pantalla; B contiene los *pixels* por byte menos uno; C contiene la máscara del *pixel.* AF y DE quedan alterados.

SCR NEXT BYTE: BC20

Halla la dirección de pantalla del punto situado 1 byte a la derecha de una dirección dada.

Condiciones de entrada: HL contiene una dirección de pantalla.

Condiciones de salida: HL contiene la dirección de pantalla actualizada. AF queda alterado.

SCR PREV BYTE: BC23

Halla la dirección de pantalla del punto situado 1 byte a la izquierda de una dirección dada.

Condiciones de entrada: HL contiene una dirección de pantalla.

Condiciones de salida: HL contiene la dirección de pantalla actualizada. AF está falseado.

SCR NEXT LINE: BC26

Halla la dirección de pantalla del punto situado una línea por debajo de una dirección dada.

Condiciones de entrada: HL contiene la dirección de pantalla dada.

Condiciones de salida: HL contiene la dirección de pantalla actualizada. AF queda alterado.

SCR PREV LINE: BC29

Halla la dirección de pantalla del punto situado una línea por encima de una dirección dada.

Condiciones de entrada: HL contiene la dirección de pantalla.

Condiciones de salida: HL contiene la dirección de pantalla actualizada. AF queda alterado.

SCR INK ENCODE: BC2C

Codifica una tinta para todos los *pixels* de un byte.

Condiciones de entrada: A contiene el número de la tinta.

Condiciones de salida: A contiene la tinta codificada. Los indicadores quedan alterados.

SCR INK DECODE: BC2F

Decodifica una tinta.

Condiciones de entrada: A contiene una máscara de tinta codificada.

Condiciones de salida: A contiene el número correspondiente de tinta. Los indicadores quedan alterados.

SCR SET INK: BC32

Fija los colores relacionados con un número de tinta. Si los dos colores son distintos, aparecen alternativamente en la pantalla.

Condiciones de entrada: A contiene el número de la tinta; B contiene el primer color; C contiene el segundo.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

SCR GET INK: BC35

Comprueba los colores relacionados con una tinta dada.

Condiciones de entrada: A contiene el número de la tinta.

Condiciones de salida: B contiene el primer color; C contiene el segundo. AF, DE y HL quedan alterados.

SCR SET BORDER: BC38

Fija los colores del borde de la pantalla. Si los colores son distintos, aparecen alternativamente.

Condiciones de entrada: B contiene el primer color; C contiene el segundo.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

SCR GET BORDER: BC3B

Comprueba los colores del borde de la pantalla.

No hay condiciones de entrada.

Condiciones de salida: B contiene el primer color; C contiene el segundo. AF, DE y HL quedan alterados.

SCR SET FLASHING: BC3E

Determina los períodos de *flash*. (En períodos de barrido.)

Condiciones de entrada: H contiene el período del primer color y L contiene el período del segundo.

Condiciones de salida: AF y HL quedan alterados.

SCR GET FLASHING: BC41

Comprueba los períodos de *flash*. (En períodos de barrido.)

No hay condiciones de entrada.

Condiciones de salida: H contiene el período del primer color; L contiene el período del segundo. AF queda falseado.

SCR FILL BOX: BC44

Rellena un área rectangular de la pantalla con una tinta dada.

Condiciones de entrada: A contiene la tinta codificada que va a utilizarse; H, la columna más a la izquierda; L, la fila superior; D, la columna más a la derecha, y E contiene la fila inferior, que van a ser rellenadas.

Las coordenadas van de 0 en adelante.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

Nota: No se tiene en cuenta el modo de escritura actual de la unidad de visualización.

SCR FLOOD BOX: BC47

Rellena un área rectangular de la pantalla con una tinta dada, hasta unos bytes que marcan los límites.

Condiciones de entrada: HL contiene la dirección de la esquina superior izquierda; D, el ancho del rectángulo (en bytes); E, la altura del rectángulo (en líneas de pantalla), y C contiene la tinta codificada que se va a utilizar.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

Nota: No se tiene en cuenta el modo de escritura actual de la unidad de visualización.

SCR CHAR INVERT: BC4A

Convierte el carácter de una posición dada a la forma de video inverso, cambiando las tintas.

Condiciones de entrada: B y C contienen las tintas codificadas; H define la columna de texto, y L la fila. (De 0 en adelante.)

Condiciones de salida: AF, BC, DE y HL quedan alterados.

SCR HW ROLL: BC4D

Desplaza la pantalla verticalmente de ocho en ocho líneas de *pixels*. La fila que aparece en la pantalla está limpia. El desplazamiento de texto no varía.

Condiciones de entrada: para un desplazamiento hacia abajo, B contiene 0; un valor distinto de 0 determina un desplazamiento hacia arriba. A contiene la tinta codificada con que aparece la nueva línea.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

SCR SW ROLL: BC50

Desplaza un área definida de pantalla de ocho en ocho líneas de *pixels*. La fila que aparece está limpia.

Condiciones de entrada: para un desplazamiento hacia abajo, B contiene 0; un valor distinto de 0 determina un desplazamiento hacia arriba. A contiene la tinta codificada con la que aparece la nueva línea; H, la columna izquierda del área a desplazar; D, la columna derecha; L, la fila superior, y E contiene la fila inferior. (Las coordenadas van de 0 en adelante.)

Condiciones de salida: AF, BC, DE y HL quedan alterados.

SCR UNPACK: BC53

Convierte la configuración de un carácter en un conjunto de máscaras de *pixel* para el modo de pantalla actual.

Condiciones de entrada: HL contiene la dirección de la configuración

del carácter; DE contiene la dirección del área en que se va a almacenar el resultado.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

SCR REPACK: BC56

Convierte un carácter en pantalla a la forma del carácter estándar.

Condiciones de entrada: A contiene la tinta codificada del carácter; H, la columna donde está el carácter, y L contiene la fila. (Las coordenadas van de 0 en adelante.) DE contiene la dirección del área donde se va a almacenar la conversión.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

SCR ACCESS: BC59

Fija el modo de gráficos de la unidad de visualización.

Condiciones de entrada: A contiene la clave del modo requerido:

0: Modo Absoluto (modo normal).

1: Modo XOR ("o" exclusivo lógico).

2: Modo AND ("y" lógico).

3: Modo OR ("o" lógico).

Condiciones de salida: AF, BC, DE y HL quedan alterados.

SCR PIXELS: BC5C

Pinta un *pixel* en la pantalla ignorando el modo de gráficos de la unidad de visualización.

Condiciones de entrada: B contiene la tinta codificada; C, la máscara del *pixel*, y HL contiene la dirección de pantalla.

Condiciones de salida: AF queda alterado.

SCR HORIZONTAL: BC5F

Dibuja una línea horizontal.

Condiciones de entrada: A contiene la tinta codificada; BC, la coordenada X del final de la línea; DE, la coordenada X del principio de la línea, y HL contiene la coordenada Y. (DE no debe ser mayor que BC.)

Condiciones de salida: AF, BC, DE y HL quedan alterados.

SCR VERTICAL: BC62

Dibuja una línea vertical.

Condiciones de entrada: A contiene la tinta codificada; BC, la coordenada Y del final de la línea: DE, la coordenada X, y HL contiene la coordenada Y del principio de la línea. (HL no debe ser mayor que BC.)

Condiciones de salida: AF, BC, DE y HL quedan alterados.

CAS INITIALISE: BC65

Inicializa el controlador del cassette, cierra las vías *(streams)*, fija la velocidad de escritura por omisión y activa los mensajes de ayuda.

No hay condiciones de entrada.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

CAS SET SPEED: BC68

Define la velocidad de escritura del cassette y la precompensación.

Condiciones de entrada: A contiene la precompensación; HL, define el período de escritura. La precompensación puede valer entre 0 y 225 (microsegundos), pero los valores altos no son los apropiados. El contenido de HL determina la duración de medio período del ciclo de un bit 0 en microsegundos. La fórmula que determina la velocidad de escritura en baudios es: 10↑6/(3*HL). HL debe valer entre 130 y 480.

Los valores estándar son: 2.000 baudios: HL = 167, A = 50. 1.000 baudios: HL = 333, A = 25.

CAS NOISY: BC6B

Controla los mensajes de ayuda del cassette.

Condiciones de entrada: A contiene 0 para habilitar los mensajes y un valor distinto de 0 para deshabilitarlos.

Condiciones de salida: AF queda alterado.

CAS START MOTOR: BC6E

Enciende el motor del cassette y espera a que se estabilice la velocidad.

No hay condiciones de entrada.

Condiciones de salida: A contiene el estado previo del motor. El acarreo queda en alto, a menos que la acción haya sido abortada pulsando Escape.

CAS STOP MOTOR: BC71

Apaga el motor del cassette.

No hay condiciones de entrada.

Condiciones de salida: A contiene el estado previo del motor. El acarreo queda en alto, salvo si se pulsó Escape.

CAS RESTORE MOTOR: BC74

Restaura el estado previo del motor.

Condiciones de entrada: A contiene el estado previo del motor.

Condiciones de salida: el acarreo queda en alto, excepto si se pulsó Escape.

CAS IN OPEN: BC77

Abre un fichero de entrada del cassette.

Condiciones de entrada: B contiene la longitud del nombre del fichero; HL contiene la dirección del nombre, y DE contiene la dirección de un área de 2K que actúa como *buffer*.

Condiciones de salida: si fue abierto correctamente, el acarreo queda en alto y el indicador de cero queda en bajo. HL contiene la dirección del *buffer* en el que está la cabecera del fichero; DE contiene la posición de los datos facilitada por la cabecera; BC, la longitud del fichero facilitada en la cabecera, y A contiene el tipo de fichero.

Si la vía *(stream)* se está utilizando, el acarreo y el indicador de cero quedan en bajo. A, BC, DE y HL quedan alterados.

Si se pulsó Escape, el acarreo y el indicador de 0 queda en alto. A, BC, DE y HL quedan alterados.

En cualquier caso, IX y los demás indicadores quedan alterados.

CAS IN CLOSE: BC7A

Cierra un fichero de entrada del cassette.

No hay condiciones de entrada.

Condiciones de salida: el acarreo queda en alto si la acción ha sido terminada, y en bajo si la vía no estaba abierta. A, BC, DE, HL y los demás indicadores quedan alterados.

CAS IN ABANDON: BC7D

Abandona un fichero de entrada del cassette.

No hay condiciones de entrada.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

CAS IN CHAR: BC80

Lee un carácter del fichero de entrada del cassette.

No hay condiciones de entrada.

Condiciones de salida: si se ejecutó la acción, A contiene el código del carácter, el acarreo queda en alto y el indicador de cero es bajo.

Si se encontró el final de fichero, A queda alterado, el acarreo queda en alto y el indicador de cero en bajo.

Si se pulsó Escape, A queda alterado, el acarreo queda en bajo y el indicador de cero en alto.

En cualquier caso, IX y los demás indicadores quedan alterados.

CAS IN DIRECT: BC83

Lee un fichero completo para almacenarlo.

Condiciones de entrada: HL contiene la dirección donde se almacenará el bloque traído desde el fichero de entrada del cassette.

Condiciones de salida: si la acción ha sido completada, HL queda en alto, y el indicador de cero en bajo.

Si el fichero no fue abierto, HL queda alterado, el acarreo y el indicador de cero quedan en bajo.

Si se pulsó Escape, HL queda alterado, el acarreo queda en bajo y el indicador de cero en alto.

En cualquier caso, A, BC, DE, IX y los demás indicadores quedan alterados.

CAS RETURN: BC86

Devuelve el último carácter leído al fichero de entrada del cassette.

No hay condiciones de entrada.

Condiciones de salida: todos los registros e indicadores son preservados.

CAS TEST EOF: BC89

Comprueba si se ha alcanzado el fin de fichero.

No hay condiciones de entrada.

Condiciones de salida: si no se ha llegado al fin de fichero, el acarreo queda en alto y el indicador de cero en bajo.

Si se ha alcanzado, el acarreo y el indicador de cero quedan en bajo.

Si se pulsa Escape, el acarreo queda en bajo y el indicador de cero en alto.

En cualquier caso A, IX y los demás indicadores quedan alterados.

CAS OUT OPEN: BC8C

Abre un fichero de salida del cassette.

Condiciones de entrada: B contiene la longitud del nombre del fichero; HL contiene la dirección del nombre, y DE contiene la dirección del área de 2K que actúa como *buffer*.

Condiciones de salida: si la vía *(stream)* ya fue abierta, el acarreo queda en bajo y HL queda alterado.

Si se realiza correctamente la apertura, el acarreo queda en alto y HL contiene la dirección del *buffer* de la cabecera.

En cualquier caso, el indicador de cero queda en bajo. A, BC, DE y IX quedan alterados.

CAS OUT CLOSE: BC8F

Cierra un fichero de salida del cassette (y lo graba en cinta).

No hay condiciones de entrada.

Condiciones de salida: si se realiza correctamente, el acarreo queda en alto y el indicador de cero en bajo.

Si el fichero no fue abierto, el acarreo y el indicador de cero quedan en bajo.

Si se pulsó Escape, el acarreo queda en bajo y el indicador de cero en alto.

En cualquier caso, A, BC, DE, HL YX y los demás indicadores quedan alterados.

CAS OUT ABANDON: BC92

Abandona un fichero de salida del cassette.

No hay condiciones de entrada.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

CAS OUT CHAR: BC95

Escribe un carácter en el fichero de salida del cassette.

Condiciones de entrada: A contiene el carácter a escribir.

Condiciones de salida: si se realiza correctamente, el acarreo queda en alto y el indicador de cero en bajo.

Si el fichero no fue abierto, el acarreo y el indicador de cero quedan en bajo.

Si se pulsa Escape, el acarreo queda en bajo y el indicador de cero en alto.

En cualquier caso, A, IX y los demás indicadores quedan alterados.

CAS OUT DIRECT: BC98

Escribe un fichero completo.

Condiciones de entrada: HL contiene la dirección de los datos a escribir; DE contiene la longitud de los datos; BC, contiene la dirección de entrada a colocar en la cabecera, y A contiene el tipo de fichero, a colocar también en la cabecera.

Condiciones de salida: si la acción se ejecuta, el acarreo queda en alto y el indicador de cero en bajo.

Si se pulsó Escape, el acarreo queda en bajo y en alto el cero.

En cualquier caso, A, BC, DE, HL, IX y los demás indicadores quedan alterados.

CAS CATALOG: BC9B

Lista los ficheros de una cinta.

Condiciones de entrada: DE contiene la dirección del área de 2K que actúa como *buffer*.

Condiciones de salida: el acarreo queda en alto si todo fue bien; en bajo si la vía *(stream)* estaba siendo utilizada (por ejemplo, un fichero de entrada abierto). En cualquier caso, el indicador de cero queda en bajo y A, BC, DE, HL, IX y los demás indicadores quedan alterados.

CAS WRITE: BC9E

Escribe un registro en cinta.

Condiciones de entrada: HL contiene la dirección de los datos a escribir; DE contiene la longitud de los datos, y A contiene el carácter de sincronismo. (&2C para la cabecera, &16 para los datos.)

Condiciones de salida: si todo fue bien, el acarreo queda en alto y A está falseado. De otro modo, el acarreo queda en baja y A contiene un código de error:

0: Pulsado Escape.

1: Error de escritura *(overrun)*.

En cualquier caso, BC, DE, HL e IX quedan alterados.

CAS READ: BCA1

Lee un registro de la cinta.

Condiciones de entrada: HL contiene la dirección en la que se van a almacenar los datos; DE contiene la longitud de los datos, y A contiene el código de sincronismo esperado.

Condiciones de salida: si todo fue bien, el acarreo queda en alto y A queda alterado. De otro modo, el acarreo queda en bajo y A contiene un código de error:

0: Pulsado Escape.

1: Error de lectura *(overrun)*.

2: Comprobación CRC fallida.

En cualquier caso, BC, DE, HL, IX y los demás indicadores quedan alterados.

Nota: CAS READ y CAS WRITE ponen en funcionamiento y detienen el motor del cassette.

CAS CHECK: BCA4

Compara un registro en cinta con el contenido de la memoria (verifica).

Condiciones de entrada: HL contiene la dirección de los datos a comparar; DE contiene la longitud de los datos, y A contiene el carácter de sincronismo esperado.

Condiciones de salida: si la comprobación fue correcta, el acarreo queda en alto y A queda alterado. De otro modo, el acarreo queda en bajo y A contiene un código de error:

- 0: Pulsado Escape.
- 1: Error de lectura *(overrun)*.
- 2: Comprobación CRC fallida.
- 3: Datos discordantes.

En cualquier caso, BC, DE, HL, IX y los demás indicadores quedan alterados.

Nota: CAS CHECK pone en funcionamiento y detiene el motor del cassette.

SOUND RESET: BCA7

Inicializa el controlador de sonido, silenciando el *chip* de sonido y borrando todas las secuencias.

No hay condiciones de entrada.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

SOUND QUEUE: BCAA

Añade un sonido a la secuencia, si es posible.

Condiciones de entrada: HL señala a un bloque de 9 bytes que definen el sonido:

Byte 0: Bit 0: Canal A.
Bit 1: Canal B.
Bit 2: Canal C.
Bit 3: Simultáneo con el canal A.
Bit 4: Simultáneo con el canal B.
Bit 5: Simultáneo con el canal C.
Bit 6: Retención del sonido hasta que sea liberado *(hold)*.
Bit 7: Secuencia inmediata.

Byte 1: Selección de amplitud de la envolvente.

Byte 2: Selección de tono de la envolvente.

Byte 3:
Byte 4: } Período del tono.

Byte 5: Período del ruido.

Byte 6: Amplitud inicial.

Byte 7:
Byte 8: } Duración.

Condiciones de salida: si se ha realizado correctamente, el acarreo queda en alto y HL queda alterado; si no, el acarreo queda en bajo y HL es preservado. A, BC, DE, IX y los demás indicadores quedan alterados.

SOUND CHECK: BCAD

Averigua si hay espacio en una secuencia de sonido.

Condiciones de entrada: A contiene el bit indicativo del canal a examinar. (Véase SOUND QUEUE anterior.)

Condiciones de salida: A contiene el estado del canal, del siguiente modo:

Bits 0-2: Número de huecos libres en la secuencia.

Bit 3: Espera para tocar simultáneamente con el canal A.

Bit 4: Espera para tocar simultáneamente con el canal B.

Bit 5: Espera para tocar simultáneamente con el canal C.

Bit 6: Retención.

Bit: 7 Activo (está produciendo sonido).

BC, DE, HL y los indicadores quedan alterados.

SOUND ARM EVENT: BCB0

Determina un suceso a ejecutar cuando la secuencia de sonido está vacía.

Condiciones de entrada: A contiene el bit indicativo del canal (véase SOUND QUEUE antes explicado) y HL contiene la dirección del bloque del suceso a ejecutar.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

(Véase KL INIT EVENT.)

SOUND RELEASE: BCB3

Libera un sonido retenido.

Condiciones de entrada: A contiene el(los) bit(s) indicadores de canal. (Véase SOUND QUEUE, explicado anteriormente.)

Condiciones de salida: AF, BC, DE, HL y IX quedan alterados.

SOUND HOLD: BCB6

Mantiene todos los sonidos a la vez.

No hay condiciones de entrada.

Condiciones de salida: el acarreo queda en alto si el sonido estaba activo. A, BC, DE, HL y los demás indicadores quedan alterados.

SOUND CONTINUE: BCB9

Continúan los sonidos que estaban retenidos.

No hay condiciones de entrada.

Condiciones de salida: AF, BC, DE, HL y IX quedan alterados.

SOUND AMPL ENVELOPE: BCBC

Determina la amplitud de una envolvente.

Condiciones de entrada: A contiene el número de la envolvente; HL, contiene la dirección del bloque de datos formado del siguiente modo:

Byte 0:	Número de secciones.
Bytes 1-3:	Primera sección.
Bytes 4-6:	Segunda sección.
Bytes 7-9:	Tercera sección.
Bytes 10-12:	Cuarta sección.
Bytes 13-15:	Quinta sección.

Dentro de cada sección, el primer byte determina la cuenta de pasos; el segundo determina la altura de cada paso, y el tercero determina el tiempo de pausa. Las secciones que no se utilicen no necesitan ser definidas. El bloque de datos no debe hallarse en un área de RAM con ROM solapada.

Condiciones de salida: el acarreo queda en alto si se ha realizado correctamente. HL contiene la dirección del bloque de datos más &10. A y BC quedan alterados.

Nota: Los detalles de esta llamada y de la siguiente son muy complejos, pero vienen referidos en el *Manual del usuario.*

SOUND TONE ENVELOPE: BCBF

Define el tono de una envolvente.

Condiciones de entrada: son análogos a los de la rutina anterior (SOUND AMPL ENVELOPE), excepto que la altura del paso se refiere al tono en lugar de a la amplitud.

Condiciones de salida: las mismas que en SOUND AMPL ENVELOPE.

SOUND A ADDRESS: BCC2

Encuentra la dirección del bloque de datos de la amplitud de una envolvente.

Condiciones de entrada: A contiene el número de la envolvente.

Condiciones de salida: Si la acción de la rutina ha sido correcta, el acarreo queda en alto, HL contiene la dirección del bloque de datos y BC contiene la longitud de la envolvente. Si la acción de la rutina ha fracasado, el acarreo queda en bajo, HL queda alterado, y BC es preservado. En cualquier caso, A queda alterado.

SOUND T ADDRESS: BCC5

Encuentra la dirección del bloque de datos del tono de una envolvente.

Los datos de entrada y de salida son igual que en la rutina anterior (SOUND A ADDRESS).

KL CHOKE OFF: BCC8

Borra todas las secuencias de sucesos y las listas del temporizador y del retorno de barrido, así como todos los sucesos síncronos pendientes y todas las funciones relacionadas con el tiempo, excepto la exploración del teclado y la generación de sonido. (Prepara el sistema para la rutina MC BOOT PROGRAM, BD13.)

No hay condiciones de entrada.

Condiciones de salida: B contiene la dirección de selección de la ROM preferente; C, la dirección de selección de ROM para un programa principal en RAM, y DE contiene la dirección de entrada para la ROM preferente actual. AF y HL quedan alterados.

KL ROM WALK: BCCB

Identifica e inicializa todas las ROM accesorias.

Condiciones de entrada: DE, contiene la dirección del primer byte utilizable en memoria; HL, del último byte utilizable en memoria.

Condiciones de salida: DE contiene la nueva dirección del primer byte utilizable y HL contiene la nueva dirección del último byte utilizable. AF y BC quedan alterados.

KL INIT BACK: BCCE

Inicializa una ROM accesoria.

Condiciones de entrada: C, contiene la dirección de selección de la ROM; DE, la dirección del primer byte utilizable en memoria, y HL contiene la dirección del último byte utilizable en memoria.

Condiciones de salida: DE contiene la nueva dirección del primer byte utilizable y HL contiene la nueva dirección del último byte utilizable. AF y B quedan alterados.

KL LOG EXT: BCD1

Crea una extensión del sistema residente. (Añadido a la lista de instrucciones externas.)

Condiciones de entrada: BC contiene la dirección de la tabla de instrucciones de extensiones del sistema residente y HL contiene la dirección de un área de RAM de 4 bytes.

Condiciones de salida: DE queda alterado.

Nota: Ni la tabla de instrucciones, ni el área de cuatro, pueden estar en zona solapada con ROM. El uso de exteriores de sistema residente se discutirá en la última parte del libro.

KL FIND COMMAND: BCD4

Busca una instrucción.

Condiciones de entrada: HL contiene la dirección en la que está almacenada la instrucción a buscar.

Condiciones de salida: si se encontró la instrucción, el acarreo queda en alto y C contiene la dirección de selección de la ROM; HL contiene la dirección de la rutina asociada. Si no se encontró la instrucción, el acarreo queda en bajo, C y HL quedan alterados. En cualquier caso, A, B y DE quedan alterados.

Nota: Sólo dieciséis letras de la instrucción son significativas.

KL NEW FRAME FLY: BCDA

Inicializa y añade un bloque a la lista de retorno del haz.

Condiciones de entrada: HL contiene la dirección del bloque; B contiene la clase de suceso; C contiene la dirección de selección de ROM para la rutina del suceso, y DE contiene la dirección de la rutina del suceso.

Condiciones de salida: AF, DE y HL quedan alterados.

KL ADD FRAME FLY: BCDA

Añade un bloque a la lista del retorno de cuadro *(frame flyback).*

Condiciones de entrada: HL contiene la dirección del bloque.

Condiciones de salida: AF, DE y HL quedan alterados.

KL DEL FRAME FLY: BCDD

Retira un bloque de la lista del retorno de cuadro.

Condiciones de entrada: HL contiene la dirección del bloque.

Condiciones de salida: AF, DE y HL quedan alterados.

KL NEW FAST TICKER: BCE0

Inicializa y añade un bloque a la lista del marcador rápido *(fast ticker).*

Condiciones de entrada: HL contiene la dirección del bloque; B contiene la clase de suceso; C contiene la dirección de selección de ROM para la rutina del suceso, y DE contiene la dirección de la rutina del suceso.

Condiciones de salida: AF, DE y HL quedan alterados.

KL ADD FAST TICKER: BCE3

Añade un bloque a la lista del marcador rápido *(fast ticker).*

Condiciones de entrada: HL contiene la dirección del bloque.

Condiciones de salida: AF, DE y HL quedan alterados.

KL DEL FAST TICKER: BCE6

Retira un bloque de la lista del marcador rápido *(fast ticker).*

Condiciones de entrada: HL contiene la dirección del bloque.

Condiciones de salida: AF, DE y HL quedan alterados.

KL ADD TICKER: BCE9

Añade un bloque a la lista del marcador *(ticker).*

Condiciones de entrada: HL contiene la dirección del bloque; DE contiene el valor inicial de la cuenta, y BC contiene el valor de recarga.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

KL DEL TICKER: BCEC

Retira un bloque de la lista del marcador *(ticker).*

Condiciones de entrada: HL contiene la dirección del bloque.

Condiciones de salida: si se ha encontrado el bloque, el acarreo queda en alto y DE contiene la cuenta restante; si no, el acarreo queda en bajo y DE queda alterado. En cualquier caso, A, HL y los demás indicadores quedan alterados.

KL INIT EVENT: BCEF

Inicializa un bloque de suceso.

Condiciones de entrada: HL contiene la dirección del bloque; B contiene la clase de suceso; C contiene la dirección de selección de ROM para el suceso, y DE contiene la dirección de la rutina del suceso.

KL EVENT: BCF2

Activa un bloque de suceso.

Condiciones de entrada: HL contiene la dirección del bloque.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

KL SINC RESET: BCF5

Borra la secuencia de sucesos síncronos.

No hay condiciones de entrada.

Condiciones de salida: AF y HL quedan alterados.

KL DEL SYNCHRONOUS: BCF8

Retira un suceso síncrono de la secuencia de sucesos.

Condiciones de entrada: HL contiene la dirección del bloque.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

KL NEXT SYNC: BCFB

Toma el siguiente suceso de la secuencia.

No hay condiciones de entrada.

Condiciones de salida: si hay un suceso esperando ser procesado, el acarreo queda en alto, HL contiene la dirección del bloque del suceso y A contiene la prioridad del suceso previo, si lo había; si no, el acarreo queda en bajo. A y HL quedan alterados. En cualquier caso, DE queda alterado.

KL DO SINC: BCFE

Ejecuta la rutina de un suceso.

Condiciones de entrada: HL contiene la dirección del bloque del suceso.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

KL DONE SYNC: BD01

Concluye el procesado de un suceso.

Condiciones de entrada: C contiene la prioridad del suceso previo; HL contiene la dirección del bloque del suceso.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

KL EVENT DISABLE: BD04

Deshabilita los sucesos síncronos normales.

No hay condiciones de entrada.

Condiciones de salida: HL queda alterado.

KL EVENT ENABLE: BD07

Habilita los sucesos síncronos normales.

No hay condiciones de entrada.

Condiciones de salida: HL queda alterado.

KL DISARM EVENT: BD0A

Impide que ocurra un suceso.

Condiciones de entrada: HL contiene la dirección del bloque del suceso.

Condiciones de salida: AF queda alterado.

KL TIME PLEASE: BD0D

Averigua el tiempo transcurrido.

No hay condiciones de entrada.

Condiciones de salida: DE/HL contienen los 4 bytes de la cuenta de tiempo; DE contiene los 2 bytes más significativos.

KL TIME SET: BD10

Fija la cuenta de tiempo.

Condiciones de entrada: DE/HL contienen los 4 bytes de la cuenta de tiempo; DE contiene los 2 bytes más significativos.

Condiciones de salida: AF queda alterado.

MC BOOT PROGRAM: BD13

Carga y ejecuta un programa.

Condiciones de entrada: HL contiene la dirección de la rutina que se llamará para cargar el programa.

No existe una salida como tal, pero si falla la carga la rutina cargadora debe volver con el acarreo en bajo. Para una carga correcta la rutina cargadora debe volver con el acarreo en alto y HL conteniendo el enlace de entrada al programa.

MC START PROGRAM: BD16

Ejecuta un programa principal.

Condiciones de entrada: HL contiene la dirección de comienzo y C contiene el byte de selección de ROM.

No existe una salida como tal.

MC WAIT FLYBACK: BD19

Espera al retorno de cuadro.

No hay condiciones de entrada.

Condiciones de salida: todos los registros e indicadores son preservados.

MC SET MODE: BD1C

Fija el modo de pantalla.

Condiciones de entrada: A contiene el modo requerido.

Condiciones de salida: AF queda alterado.

MC SCREEN OFFSET: BD1F

Determina el desplazamiento de pantalla.

Condiciones de entrada: A contiene la base de pantalla requerida; HL contiene el desplazamiento requerido.

Contenido de salida: AF queda alterado.

MC CLEAR INKS: BD22

Pone todas las tintas y el borde del mismo color.

Condiciones de entrada: DE contiene un vector de tinta; D contiene el color común de la tinta, y E contiene el color del borde.

Condiciones de salida: AF queda alterado.

MC SET INKS: BD25

Fija el color de todas las tintas.

Condiciones de entrada: DE contiene la dirección de un bloque de definición de tintas, en el que el byte 0 define el color del borde, y los bytes del 1 al 16 definen el color de las tintas de 0 a 15.

Condiciones de salida: AF queda alterado.

MC RESET PRINTER: BD28

Restaura la dirección normal de entrada de la impresora.

No hay condiciones de entrada.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

MC PRINT CHAR: BD2B

Manda un carácter al puerto de la impresora.

Condiciones de entrada: A contiene el carácter a mandar. (El bit 7 será ignorado.)

Condiciones de salida: el acarreo queda en alto si la acción ha sido completada, y en bajo si se acabó el tiempo de espera (aproximadamente, 0,4 segundos) sin completar la acción. En cualquier caso, A y los demás indicadores quedan alterados.

MC BUSY PRINTER: BD2E

Comprueba si el puerto de la impresora está ocupado.

No hay condiciones de entrada.

Condiciones de salida: el acarreo queda en alto si el puerto está ocupado, y en bajo si no lo está. Los demás indicadores quedan alterados.

MC SEND PRINTER: BD31

Manda un carácter al puerto de la impresora. (No comprueba si está ocupada.)

Condiciones de entrada: A contiene el carácter a mandar. (El bit 7 se ignora.)

Condiciones de salida: el acarreo queda en alto y A queda alterado.

MC SOUND REGISTER: BD34

Manda datos al *chip* de sonido.

Condiciones de entrada: A contiene el número del registro; C contiene el byte de datos.

Condiciones de salida: AF y BC quedan alterados.

JUMP RESTORES: BD37

Restablece la tabla de saltos estándar.

No hay condiciones de entrada.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

Con esto terminamos la tabla de saltos *(jumpblock)* del *firmware* principal. Seguidamente se exponen los saltos indirectos del *firmware*, insertándose una continuación de la tabla de saltos principal para la que no existen definiciones. (Funciones del BASIC.)

Indirecciones (saltos indirectos al "firmware" o sistema operativo)

IND:TXT UNDRAW CURSOR: BDD0

Retira el cursor de la pantalla.

No hay condiciones de entrada.

Condiciones de salida: AF queda alterado.

IND:TXT WRITE CHAR: BDD3

Escribe un carácter en pantalla.

Condiciones de entrada: A contiene el código del carácter; H contiene la columna, y L contiene la fila. (Las coordenadas van de 0 en adelante.)

Condiciones de salida: AF, BC, DE y HL quedan alterados.

IND:TXT UNWRITE: BDD6

Lee un carácter de la pantalla.

Condiciones de entrada: H contiene la columna y L contiene la fila donde se va a leer el carácter.

Condiciones de salida: si se reconoce el carácter, el acarreo queda en alto y A contiene el código del carácter; si no, el acarreo queda en bajo y A contiene 0. En cualquier caso, BC, DE, HL y los demás indicadores quedan alterados.

IND:TXT OUT ACTION: BDD9

Manda un carácter o un código de control.

Condiciones de entrada: A contiene el carácter o el código de control.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

IND:GRA PLOT: BDDC

Marca un punto.

Condiciones de entrada: DE contiene la coordenada X y HL contiene la coordenada Y.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

IND:GRA TEST: BDDF

Comprueba un punto.

Condiciones de entrada: DE contiene la coordenada X y HL contiene la coordenada Y.

Condiciones de salida: A contiene la tinta decodificadora del punto. BC, DE, HL y los indicadores quedan alterados.

IND:GRA LINE: BDE2

Dibuja una línea desde la posición del cursor de gráficos.

Condiciones de entrada: DE contiene la coordenada X y HL contiene la coordenada Y del punto de llegada.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

IND:SCR READ: BDE5

Lee un punto de la pantalla y decodifica su tinta.

Condiciones de entrada: HL contiene la dirección de pantalla del punto y C contiene la máscara del punto.

Condiciones de salida: A contiene la tinta del punto decodificada. Los indicadores quedan alterados.

IND:SCR WRITE: BDE8

Activa uno o varios puntos en la pantalla.

Condiciones de entrada: HL contiene la dirección de pantalla del(de los) punto(s); C contiene la máscara del punto, y B contiene la tinta codificada a utilizar.

Condiciones de salida: AF queda alterado.

IND:SCR MODE CLEAR: BDEB

Borra la pantalla con la tinta 0.

No hay condiciones de entrada.

Condiciones de salida: AF, BC, DE y HL quedan alterados.

IND:KM TEST BREAK: BDEE

Comprueba si ha ocurrido un BREAK o un RESET, llamando al suceso apropiado.

Condiciones de entrada: las interrupciones deben estar deshabilitadas; C debe contener el estado de las teclas SHIFT y CTRL.

Condiciones de salida: AF y HL quedan alterados.

IND:MC WAIT PRINTER: BDF1

Escribe un carácter o ejecuta un tiempo de espera.

Condiciones de entrada: A contiene el código del carácter a mandar.

Condiciones de salida: si la acción ha sido realizada, el acarreo queda en alto; si no, el acarreo queda en bajo. A y BC quedan alterados.

Con lo expuesto se completan las indirecciones. A continuación veremos el núcleo alto de la tabla de saltos *(jumpblock)* que accede a las rutinas almacenadas en RAM.

Núcleo alto de la tabla de saltos (High Kernel Jumpblock)

HI:KL U ROM ENABLE: B900

Habilita la ROM superior.

No hay condiciones de entrada.

Condiciones de salida: A contiene el estado anterior de la ROM.

HI:KL U ROM DISABLE: B903

Deshabilita la ROM superior.

No hay condiciones de entrada.

Condiciones de salida: A contiene el estado anterior de la ROM.

HI:KL L ROM ENABLE: B906

Habilita la ROM inferior.

No hay condiciones de entrada.

Condiciones de salida: A contiene el estado anterior de la ROM.

HI:KL L ROM DISABLE: B909

Deshabilita la ROM inferior.

No hay condiciones de entrada.

Condiciones de salida: A contiene el estado anterior de la ROM.

HI:KL ROM RESTORE: B90C

Fija el estado de la ROM.

Condiciones de entrada: A contiene el estado de la ROM, como el que devolvían las rutinas anteriores.

Condiciones de salida: AF queda alterado.

HI:KL ROM SELECT: B90F

Selecciona una ROM superior.

Condiciones de entrada: C contiene la dirección de selección para la ROM requerida.

Condiciones de salida: C contiene la dirección de selección anterior; B contiene el estado anterior de la ROM. AF queda alterado.

HI:KL CURR SELECTION: B912

Comprueba que la ROM superior está activada actualmente.

No hay condiciones de entrada.

Condiciones de salida: A contiene la dirección de selección para la ROM superior actual.

HI:KL PROBE ROM: B915

Comprueba la clase y la versión de la ROM superior.

Condiciones de entrada: C contiene la dirección de selección de la ROM a probar.

Condiciones de salida: A contiene la clase de ROM; L contiene el número-marca, y H contiene el número de la versión. B y los indicadores quedan alterados.

HI:KL ROM DESELECT: B918

Devuelve la ROM superior previamente seleccionada.

Condiciones de entrada: B contiene el estado de la ROM anterior; C contiene la dirección de selección anterior.

Condiciones de salida: C contiene la dirección de selección de la ROM aplicable ahora. B queda alterado.

HI:KL LDIR: B91B

Copia un bloque de memoria con un puntero incremental, con las ROM deshabilitadas.

Condiciones de entrada: BC contiene la longitud del bloque; DE contiene la dirección de destino, y HL contiene la dirección de comienzo. (Valores iniciales.)

Condiciones de salida: F, BC, DE y HL están como si hubieran sido tratados por una instrucción LDIR.

Nota: No hay que utilizar esta llamada, si el área original y el área de destino están solapadas.

HI:KL LDDR: B91E

Es como LDIR, pero con un puntero decremental. Se puede utilizar cuando el uso de LDIR sea inapropiado, como se indicó anteriormente.

HI:KL POLL SYNCHRONOUS: B921

Comprueba si está pendiente un suceso de mayor prioridad que el suceso actual.

No hay condiciones de entrada.

Condiciones de salida: el acarreo queda en alto si está pendiente un suceso de mayor prioridad. A y los demás indicadores, quedan alterados.

Nota: Esta última entrada es directa y no puede ser modificada.

PARTE SEGUNDA

EL INTERFAZ

Sonido	1	2	Masa
A15	3	4	A14
A13	5	6	A12
A11	7	8	A10
A9	9	10	A8
A7	11	12	A6
A5	13	14	A4
A3	15	16	A2
A1	17	18	A0
D7	19	20	D6
D5	21	22	D4
D3	23	24	D2
D1	25	26	D0
+5v	27	28	$\overline{\text{MREQ}}$
$\overline{\text{M1}}$	29	30	$\overline{\text{RFSH}}$
$\overline{\text{IORQ}}$	31	32	$\overline{\text{RD}}$
$\overline{\text{WR}}$	33	34	$\overline{\text{HALT}}$
$\overline{\text{INT}}$	35	36	$\overline{\text{NMI}}$
$\overline{\text{BUSRQ}}$	37	38	$\overline{\text{BUSAK}}$
READY	39	40	$\overline{\text{BUS RESET}}$
$\overline{\text{RESET}}$	41	42	$\overline{\text{ROMEN}}$
ROMDIS	43	44	$\overline{\text{RAMRD}}$
RAMDIS	45	46	CURSOR
LIGHT PEN	47	48	$\overline{\text{EXP}}$
MASA	49	50	CLOCK

Figura 2.1. Conector del puerto de expansión.


64K COLOUR PERSONAL COMPUTER

2
El interfaz

Las situaciones de los bornes del conector de expansión de cincuenta vías se muestran en la figura 2.1. Por él salen todas las líneas del procesador central, más un número de líneas de control del sistema, pero ello no quiere decir que puedan usarse todas.

Las líneas A0-A15 definen una dirección de memoria o de entrada/salida. Si la salida $\overline{\mathrm{MREQ}}$ está en bajo, se solicita un acceso a memoria. Pero si es la salida $\overline{\mathrm{IORQ}}$ la que está en bajo, entonces se solicita una transferencia de entrada/salida. La dirección debe ser interpretada de acuerdo con lo solicitado. Los datos son transferidos en cada caso por las líneas de datos bidireccionales D0-D7.

La línea $\overline{\mathrm{M1}}$ origina confusiones. Es una salida que está en bajo cuando el procesador central está cargando un código de operación —el primer byte de una instrucción—, excepto cuando el primer byte es &CB, &DD, &ED y &FD, en cuyo caso el código de operación ocupa los dos primeros bytes. $\overline{\mathrm{M1}}$ no permanece en bajo mientras se cargan cualesquiera de los siguientes bytes de la instrucción; por tal motivo no se puede usar para controlar un sistema que cargue instrucciones de un área de memoria y datos de otra.

Si $\overline{\mathrm{M1}}$ e $\overline{\mathrm{IORQ}}$ están en bajo simultáneamente, el procesador está reconociendo una interrupción.

La salida $\overline{\mathrm{RFSH}}$ está en bajo cuando las líneas de las siete

direcciones menos significativas están transmitiendo una dirección de refresco para la memoria dinámica. El recurso de refresco es uno de los puntos fuertes del procesador Z80, pero debe utilizarse con cuidado, ya que hay circunstancias en las que se suspende el servicio de refresco, como se explica a continuación.

$\overline{\text{HALT}}$ es una salida que está en bajo cuando el procesador ha ejecutado una instrucción $\overline{\text{HALT}}$ y está esperando una interrupción. Durante este período el procesador ejecuta NOP para mantener las acciones de refresco.

$\overline{\text{INT}}$ es la principal entrada de interrupción del procesador. Debe ser controlada mediante una salida en colector abierto, que, asimismo, debe estar en bajo para llamar a una interrupción. En este sistema el procesador responde saltando a la dirección 0038, mientras que la dirección de la instrucción que debería haberse ejecutado se guarda en la pila; de esta manera se puede volver a esta instrucción cuando se complete el proceso de interrupción. Se utiliza el modo de interrupción 1 del Z80.

$\overline{\text{NMI}}$ es una línea de interrupción no enmascarada. Nunca debe de estar en bajo, ya que saltaría a la dirección 0066, que puede estar en RAM o en ROM. Por tanto, los resultados pueden ser impredecibles, probablemente catastróficos.

$\overline{\text{BUSRQ}}$ es una entrada del procesador. Cuando está en bajo se pide al procesador que libere el control de todas sus líneas excepto $\overline{\text{M1}}$, $\overline{\text{RFSH}}$ y $\overline{\text{HALT}}$. Tras completar la instrucción que estuviera ejecutando el procesador libera las líneas y pone $\overline{\text{BUSAK}}$ en bajo para indicar que lo ha hecho. Ahora un sistema externo puede controlar el *bus* y reemplazar las acciones de control que normalmente ejecuta el procesador. Cuando $\overline{\text{BUSRQ}}$ está nuevamente en alto, el sistema vuelve a su estado normal.

El uso más común de este recurso es el acceso directo de memoria (DMA), que permite la transferencia de datos a y desde la memoria o canales de entrada/salida sin necesidad de ejecutar un programa. Esta transferencia puede ser muy rápida, pero no es aconsejable mantener $\overline{\text{BUSRQ}}$ en bajo durante mucho tiempo, ya que esto detiene la acción habitual de refresco. Lo normal es transferir un byte DMA cada vez. A esta operación se le llama "robo de un ciclo" *(cycle stealing)* porque el sistema externo toma el control de la máquina durante un ciclo.

READY es la señal WAIT del procesador, una entrada del procesador que hace que prolongue su ciclo de máquina insertando estados de espera, generalmente porque un periférico o un dispositivo de memoria no están preparados para la transferencia.

Con esto se completan las señales del procesador. Las salidas no son amplificadas dentro del CPC464, y no deben conectarse a más de una carga TTL. Del mismo modo, la salida de 5v no suministra más de 100 mA. (Es posible que suministre un poco más si el cassette está desconectado.)

Las líneas relacionadas con el sistema del CPC464 son las siguientes:

$\overline{\mathrm{ROMEN}}$ es una salida que se pone en bajo para seleccionar una ROM superior externa. ROMDIS es una entrada que se pone en alto para deshabilitar la ROM interna. Del mismo modo $\overline{\mathrm{RAMRD}}$ está en bajo para habilitar la RAM externa, mientras que la entrada RAMDIS se pone en alto para desactivar la RAM interna.

La temporización de estas señales es importante. Si dos fuentes de datos se conectan simultáneamente al *bus* de datos pueden "quemarse" mutuamente. ROMDIS debe ponerse en alto antes de bajar $\overline{\mathrm{ROMEN}}$, etc.

CURSOR es la señal de CURSOR del controlador del CRT, y está en alto para indicar una dirección de cursor válida en los registros 10 y 11 del controlador.

LIGHT PEN es la entrada del lápiz óptico del controlador del CRT, el cual actúa como se describió en el capítulo de las entradas.

$\overline{\mathrm{EXP}}$ es el bit 5 del puerto B del PPI. Este bit está identificado como "Reservado" en la documentación, y normalmente está en bajo. Utilizar esta línea implica hacer uso del registro C′, en el cual se almacena el estado de los otros bits del puerto B. Por consiguiente, debe ser manipulado con cuidado.

$\overline{\mathrm{BUS\ RESET}}$ se pone en alto a través de una resistencia de 2K2. Si se conecta a masa se produce una inicialización completa del sistema, como al ponerlo en funcionamiento.

SOUND se conecta a través de resistencias de 10K a las tres salidas de tono del generador de sonido.

La señal de CLOCK es la señal de reloj de 4 MHz.

Aunque estos datos se dan de buena fe, están basados en una información escasa; por consiguiente, deben utilizarse con precaución.

PARTE TERCERA

LAS SALIDAS


64k COLOUR PERSONAL COMPUTER

3
Las salidas

Principios generales

Puesto que tienes acceso a casi todas las líneas del procesador central, virtualmente no hay límite, en teoría, para la variedad de equipos externos que pueden ser conectados al CPC464. El primer paso siempre será la creación de un interfaz en paralelo, que puede hacerse para controlar otros dispositivos, o ser controlado por ellos. La transferencia de datos de entrada y salida es la clave del proceso, que puede necesitar ser soportado por el paso de datos de control para fijar los modos, asegurar un correcto sincronismo y percibir las condiciones externas.

Una vez que el interfaz en paralelo está comprobado, el dato que maneja puede ser convertido a un dato en serie. De modo similar, es posible la conversión hacia o desde datos análogos, descubriendo un rango de posibilidades completamente nuevo. Finalmente, el CPC464 permite acceder a extensiones de programa, pero ésta es una materia muy compleja. Se dará información a este respecto, pero un tratamiento completo es demasiado extenso para un libro de estas características.

Puede advertirse que las extensiones de *hardware* deben compartir las ventajas con el sistema de disco y otras extensiones AMSTRAD.

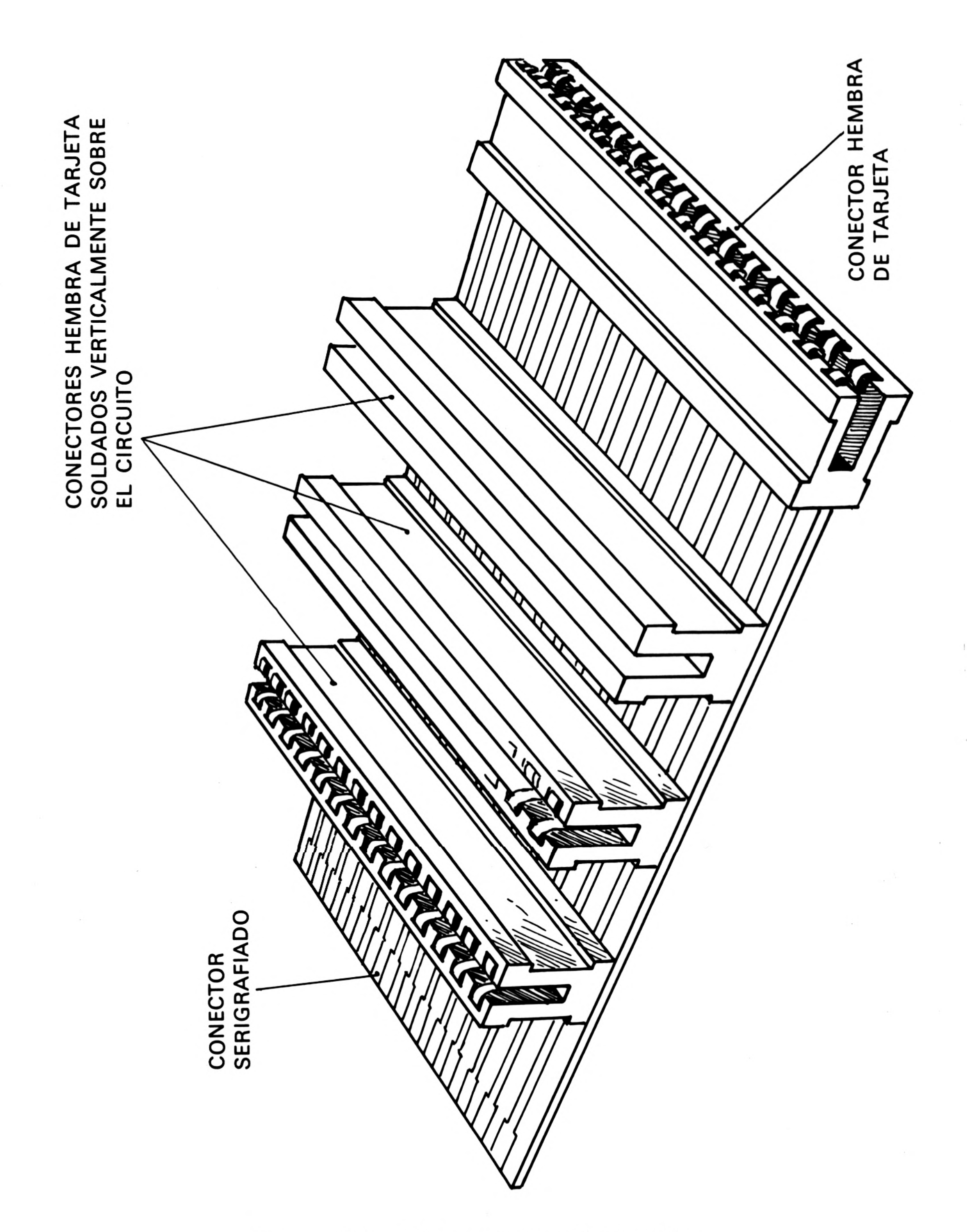


Figura 3.1. Expansor de tarjeta madre.

Por tanto, interesa utilizar un sistema de "tarjeta-madre" *(motherboard)*. Esta tarjeta se enchufa en el interfaz externo, dotándolo de peculiaridades para conectar el sistema de disco, así como otros periféricos. El esquema que se muestra en la figura 3.1 puede aclarar esta idea. La tarjeta-madre dispone de tres conectores-hembra de tarjeta que van soldados a ella verticalmente, mientras que al otro extremo lleva un conector, serigrafiado en la misma tarjeta, igual que la salida del CPC464.

Evita la tentación de trabajar desordenadamente, con cables por todas partes, y evita también utilizar racimos de cables enroscados. Las señales con las que estás trabajando tienen gran amplitud de banda y pueden producir interferencias en las líneas próximas. Los cables planos son mejores, sobre todo cuando hay toma de tierra cada dos líneas, y es especialmente bueno el tipo de pares trenzados, si lo encuentras.

Es importante vigilar la carga en las líneas del interfaz, así como en la fuente de alimentación de 5v. Donde es posible se especifican las cargas-límite, aunque no todas son conocidas suficientemente. Es aconsejable utilizar *buffers*, pero recuerda que introducen retardos de tiempo, que pueden ser críticos en algunos sistemas.

Se pueden encontrar tarjetas ya serigrafiadas, preparadas para soldar los conectores. Los agujeros para soldar los conectores son normalmente de 0,1 pulgadas (2,54 milímetros).

Se pueden obtener más detalles dirigiéndose a Vero Electronics Ltd., Industrial Estate, Chandler's Ford, Hampshire, U.K.

Interfaz en paralelo

Reglas del interfaz

En un análisis final, todos los interfaces entre el ordenador y el mundo exterior son generalmente en paralelo. La transformación a (o de) señales en serie o similares es cuestión del circuito externo. El ordenador sólo puede comunicarse en paralelo.

La excepción a esta regla surge cuando un adaptador en serie especial se adapta como una parte interna del ordenador. En el caso del CPC464, un interfaz adaptador se considera como extensión externa.

Para la salida de datos, el estado de las líneas de datos D0-D7 debe ser copiado en un conjunto de biestables, ya que en estas líneas

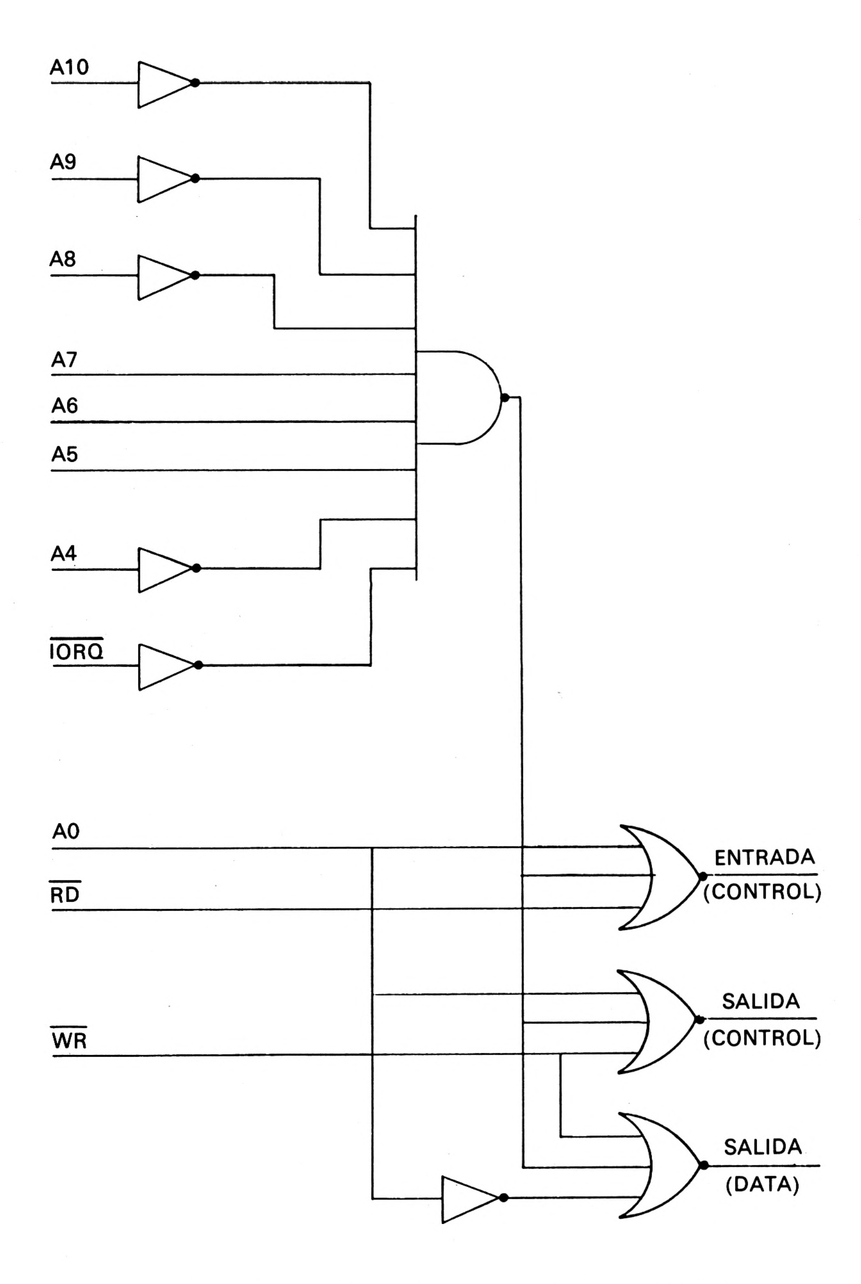


Figura 3.2. Control para un puerto en paralelo.

los datos están disponibles muy poco tiempo. La copia se realiza cuando se satisfacen las siguientes condiciones:

- La dirección entrada/salida pertinente está presente en las líneas A0-A15, aunque no es necesario tener en cuenta todas las líneas.
- La línea $\overline{IORQ}$ está en bajo, indicando una transferencia entrada/salida.
- La línea $\overline{WR}$ está en bajo, indicando una transferencia de salida (escritura).

Para la entrada de datos, las fuentes normalmente deben permitir cualquier valor en las líneas de datos. Esto requiere utilizar un dispositivo “tri-estado”, el cual puede poner en alto o en bajo las líneas, o evitar influencias de ellas, presentando una impedancia alta. Todas las fuentes, excepto una, deben tener alta impedancia en cada momento. Las condiciones que se requieren son las siguientes:

- Se identifica la dirección entrada/salida pertinente.
- La línea $\overline{IORQ}$ está en bajo.
- La línea $\overline{RD}$ está en bajo.

Como habrás observado, hay reglas que limitan la selección de direcciones para usar periféricos. Todas las direcciones permitidas tienen en bajo el bit 10, lo cual puede utilizarse para simplificar la de codificación. Supón, por ejemplo, que decides utilizar las direcciones F8E0 y F8E1: la primera para entrada y salida de datos de control, y la segunda para salida de datos. El circuito que se muestra en la figura 3.2 proporcionará las señales de control requeridas.

La puerta NAND de ocho entradas es atacada por las señales $\overline{IORQ}$, A4, A5, A6, A7, A8, A9, A10. Su salida estará en bajo cuando se genere una dirección de entrada/salida de la forma XXXX X000 1110 XXXX. Si se observan las reglas de limitación de direcciones, éstas estarán comprendidas entre F8E0-F8EF.

La salida de la puerta NAND se manda a tres puertas NOR de tres entradas. La primera, es atacada además por A0 y RD. Su salida estará en alto si la salida de la puerta NAND está en bajo, así como A0 y RD. Por tanto, esto indicará una entrada desde una dirección par en el rango antes indicado. Esto incluye a F8E0, pero también abarca a F8E2, F8E4, etc. Si van a utilizarse las otras direcciones, es necesario utilizar una codificación adicional.

La segunda puerta NOR recibe la salida de la puerta NAND, A0

y WR. Esta mandará una señal en alto, hacia una dirección par, para la transferencia de salida. Las direcciones que satisfacen estos requisitos se han indicado en el párrafo anterior.

La tercera puerta NOR recibe la salida de la puerta NAND, A0 y WR. Esta mandará una señal en alto para una transferencia de salida a una dirección impar dentro del rango F8E1-F8EF.

Las salidas de las tres puertas NOR controlan el resto del sistema entrada/salida.

Un puerto de impresora alternativo

Para ilustrar cómo se puede utilizar el circuito anterior, consideraremos el proveerse de un puerto de impresora de 8 bits completo. Necesitaremos un biestable de 8 bits para admitir salidas de datos y controlar la impresora. Se necesitará también una línea de control de salida simple para generar la señal de sincronismo *(strobe)*, que le indica a la impresora que puede leer los datos. También hará falta una línea de control de entrada simple para detectar el estado de la señal de ocupado (BUSY), que indica si la impresora está inhabilitada, por el momento, para recibir nuevos datos.

Algunos sistemas utilizan la señal ACK en lugar de la de BUSY.

Comentaremos esta particularidad: BUSY está en alto cuando la impresora acepta una entrada, y permanece así hasta que pueda aceptarse una nueva entrada. Esta señal puede permanecer en alto mientras se está imprimiendo una línea de datos almacenada, o sólo lo suficiente para que sea almacenado un único código de carácter.

ACK, en cambio, es un impulso de corta duración que dura tan sólo unos microsegundos, que coincide con la transición de alto a bajo de la señal de BUSY. Es posible que detectar este pulso resulte difícil en una transferencia de lectura. Debe utilizarse para borrar un biestable activado por $\overline{\text{STROBE}}$, simulando una forma local de BUSY.

Además, la utilización de BUSY permite detectar situaciones en las que la impresora no está conectada, en cuyo caso la línea de BUSY estará en alto todo el tiempo. Para descubrir esto, la línea de BUSY debe ser comprobada en un momento en que no se haya mandado una señal de *strobe* durante algunos segundos.

Aparentemente, la señal de control de salida de la segunda puerta NOR puede invertirse para suministrar una señal $\overline{\text{STROBE}}$. Este

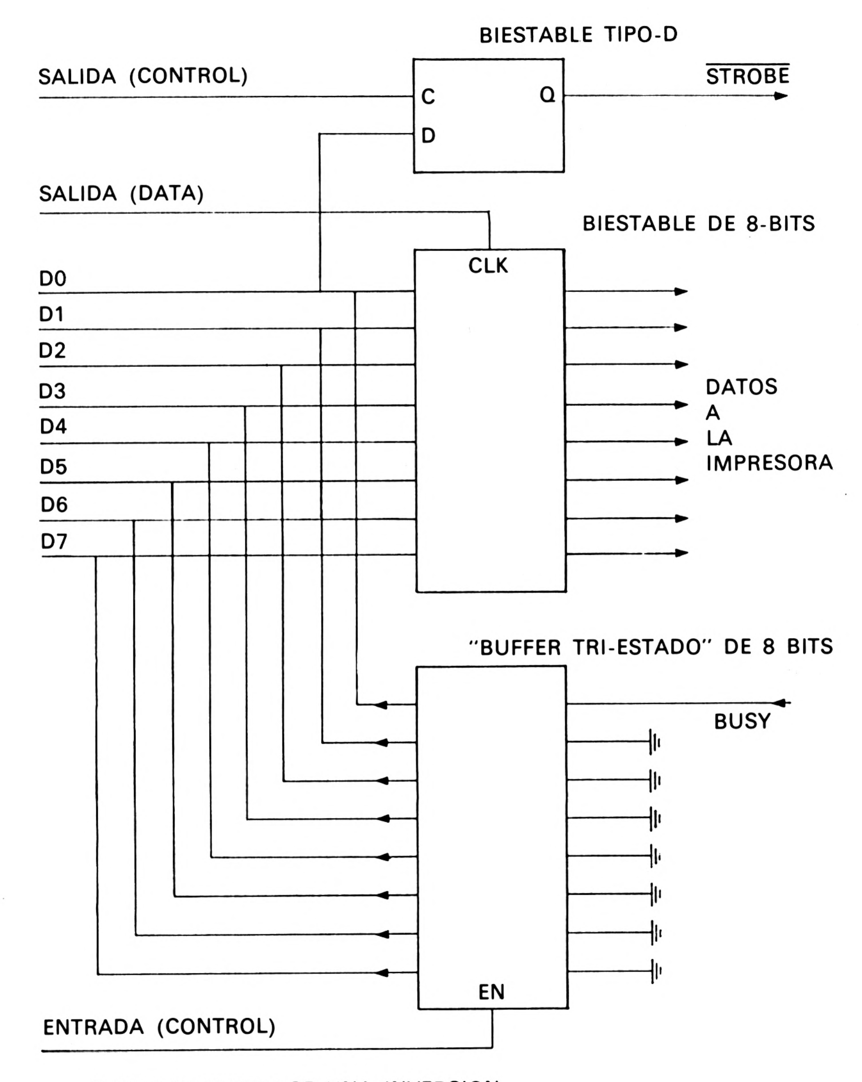


Figura 3.3. Puerto de impresora en paralelo.

método se ha utilizado con éxito, pero el impulso de STROBE no puede durar más de medio microsegundo, fracción demasiado corta para algunas impresoras. La salida de la puerta NOR debe poner un biestable a un estado determinado por la línea de datos.

Por otro lado, la señal de control de entrada debe utilizarse para habilitar un elemento con salida "tri-estado". Estos elementos no son fáciles de encontrar en forma de un solo canal; por eso, el circuito que se muestra en la figura 3.3 utiliza un elemento de 8 bits. Tal particularidad es útil porque te provee de otras líneas de control de entrada, y el control de salida puede hacerse del mismo modo utilizando, por idéntica razón, 8 bits.

Evidentemente algunas impresoras proveen líneas de salida adicionales indicando su estado interno, y también aceptan entradas adicionales que controlan dicho estado. El sistema mostrado se ajusta a lo expuesto, aunque se precisará un soporte de *software* adicional para manejar los datos adicionales.

Soporte de "software"

El *firmware* original de impresora del CPC464 no tiene utilidad en lo que concierne al puerto de impresora externo, ya que está diseñado para la distribución del puerto del *hardware* interno. Se facilita un conjunto de rutinas revisadas. Las entradas de la tabla de saltos *(jumpblock)* deben modificarse para acceder a esas rutinas. Hay cinco llamadas importantes a la impresora, y mientras una puede servir para propósitos generales, es mejor conservar el diseño original; de este modo las llamadas siguen siendo válidas. Las cinco llamadas mencionadas se resumen a continuación:

- BD2B: MC PRINT CHAR: Llama a MC WAIT PRINTER con BC preservado.
- BDF1: MC WAIT PRINTER: Entra en MC SEND PRINTER si MC BUSY PRINTER vuelve con el acarreo a cero.
- BD31: MC SEND PRINTER: Manda un byte a la impresora.
- BD2E: MC BUSY PRINTER: Vuelve con acarreo a cero si BUSY está en bajo.
- BD28: MC RESET PRINTER: Restaura las entradas a la tabla de saltos para MC WAIT PRINTER.

Las rutinas requeridas se definen aquí en ensamblador. Las posiciones en las que se van a almacenar pueden dejarse para después.

```
MPC   PUSH  BC          ;MC PRINT CHAR.
      CALL  MWP         ;Llama a MC WRITE
                          PRINTER.
      POP   BC
      RET

MWP   LD    BD,&0032    ;MC WRITE PRINTER. Fija
                          la cuenta del tiempo de
                          espera.
L1    CALL  MBP         ;Llama a MC BUSY
                          PRINTER.
      JR    NC, MSP     ;Si BUSY está en bajo hay
                          salida de datos.
      DJNZ  L1          ;Bucle de cincuenta
                          repeticiones.
      DEC   C
      JR    NZ, L1      ;Repetir 256 veces.
      OR    A           ;Borra el acarreo: la acción
                          falló en el tiempo de espera.
      RET

MSP   PUSH  BC          ;MC SEND PRINTER.
      LD    BC,&F8E1    ;Dirección del dato.
      OUT   (C),A       ;Fija el dato.
      DEC   BC          ;Dirección de control.
      XOR   A
      OUT   (C),A       ;Activa el STROBE
      LD    A,1
      OUT   (C),A       ;Desactiva el STROBE
      POP   BC
      SCF               ;Pone el acarreo en alto.
                          Acción correcta.
      RET

MBP   PUSH  BC          ;MC BUSY PRINTER.
      LD    BC,&F8E0    ;Dirección de control.
      IN    C,(C)       ;Lee BUSY.
      RR    C           ;Bit 0 al acarreo.
      POP   BC
      RET
```

```
RES     LD      A,&C3           ;Esta rutina restaura la tabla
                                  de saltos.
        LD      (&BD2B),A
        LD      HL,MPC
        LD      (&BD2C),HL
        LD      (&BDF1),A
        LD      HL,MWP
        LD      (&BD2F),HL
        LD      (&BD31),A
        LD      HL,MSP
        LD      (&BD32),HL
        LD      HL,MBP
        LD      (&BD2F),HL
        RET
```

Observa que las entradas se hacen mediante saltos directos, en lugar de la llamada &CF, que normalmente se utiliza para estos enlaces. Estas rutinas deben almacenarse en la memoria de usuario, pero no sobre las ROM; de este modo la selección del estado de la ROM es irrelevante.

Pero... ¿dónde se almacenan los programas? El BASIC utiliza cualquiera posición hasta A3FF, y además necesitará bajar esta dirección si se ha utilizado SYMBOL AFTER 0, para almacenar todos los modelos de los caracteres en RAM. Si introducimos MEMORY &A000 habremos reservado 1K de RAM protegida. Esto es importante, porque todas las posiciones desde LOMEM hasta HIME se borran cuando se carga un programa o cuando se utiliza la instrucción NEW.

No es necesario reservar un kilobyte completo para los programas que nos conciernen por el momento, pero el área reservada puede utilizarse como nosotros queramos. La llamada MEMORY &A000 puede servir como ejemplo.

Una vez restaurado el límite superior de memoria, no podrás utilizar SYMBOL AFTER, ya que los modelos de caracteres copiados podrán ser divididos, en vez de estar contiguos.

Otra advertencia; si hay conectadas ROM auxiliares, el área de almacenamiento encargada de otras funciones puede cambiar. Las rutinas para controlar la impresora deben ser relocalizables, para permitirlo. No obstante, generalmente será factible situarlas en una posición predecible.

Comunicación con ordenadores

Hay veces en que es conveniente ser capaz de pasar datos de un ordenador a otro. Si los ordenadores son del mismo tipo, la transmisión por medio de cintas de cassette puede ser oportuna. Tiempo atrás se consideró éste como el único método viable para los ordenadores personales —y también para otros tipos de ordenadores— para que se comunicaran unos con otros, pese a la necesidad de *hardware* y *software* compatibles.

Recientemente, la transmisión en serie se ha hecho más popular para este propósito, que se puede considerar como el antiguo método de cinta... pero sin cinta, aunque pueden aparecer problemas de incompatibilidad de velocidades de transmisión y de formato de los datos.

La transmisión en paralelo se evitó porque requería conexiones directas entre los ordenadores y podían resultar arriesgadas. La llegada de los opto-aisladores eliminó esta preocupación, posibilitando una transferencia de datos sobre ordenadores completamente distintos conectando al ordenador receptor un interfaz adecuado.

Los opto-aisladores consisten en un diodo emisor de luz y una célula fotosensible, montadas en una misma unidad. Una señal de entrada provoca luz en el diodo, y la llegada de la luz emitida a la célula fotosensible provoca una señal de salida, sin existir una conexión física intermedia. Se necesita un opto-aislador para cada línea de datos y para cada línea de control.

El circuito para un interfaz en paralelo de entrada es casi idéntico al interfaz en paralelo de salida. Difieren en los siguientes puntos:

- Los biestables de salida de datos se reemplazan por un *buffer* de tres estados, similar al utilizado en el interfaz de salida, para el control de entrada.
- Debe añadirse un biestable para generar la señal de BUSY. Se alza con la señal de $\overline{\text{STROBE}}$, y se borra mediante un impulso generador por el *software* de control.
- El *software* debe estar preparado para almacenar datos antes de leerlos, y mandar un impulso ACK cuando se haya completado la carga.

Debemos tener en cuenta desde las consideraciones del *hardware* hasta las implicaciones del *software*.

Si el ordenador emisor posee la facultad de generar un fichero en ASCII de un programa en BASIC, los datos resultantes pueden ser

utilizados por el ordenador receptor como si se trataran de un ingreso por teclado, aunque probablemente no sea posible correr el programa sin hacer algunos ajustes. Un programa transferido desde un ordenador BBC a un CPC464 necesitará, presumiblemente, que le sean añadidos algunos espacios; además, las instrucciones de gráficos y de sonido deberán ser cambiadas, pero esto es menos tedioso que realizar la transferencia del programa a mano.

La transferencia de datos es generalmente directa, pero el código máquina sólo es transferible entre ordenadores que utilicen el mismo procesador. En teoría, es posible convertir los códigos de un procesador al código máquina de un procesador distinto, pero el traductor puede dejar memoria insuficiente para los demás. Los emuladores que ejecutan códigos máquina de otros procesadores constituyen un tema distinto, y son bastante lentos.

La transferencia de datos de un ordenador a otro no es todo. Una vez transferidos, probablemente será necesaria alguna clase de conversión para poner el código o el dato de una forma inteligible.

Corremos el peligro de perdernos en un área de gran complejidad. Basta decir que la idea de la transferencia de datos entre ordenadores distintos no es imposible, como se ha podido comprobar. Aquellos que deseen experimentar en esta área deben reflexionar sobre la idea de utilizar una tabla de saltos *(jumpblock)* intermedia para transformar las entradas del sistema operativo y acepten el código "extraño"...

Interfaz en serie

Un interfaz en serie se utiliza cuando se precisa mandar datos a una velocidad limitada con tan sólo un par de cables. La transmisión en paralelo es más rápida y más simple, pero los numerosos cables necesarios entrañan algunos inconvenientes.

Los interfaces en serie pueden tomar varias formas, pero hoy en día lo normal es transmitir los datos en bytes separados, mejor que de una forma continua, ya que permite eliminar la transmisión de una señal de reloj.

El sistema trabaja con señales de reloj distintas a cada extremo de la línea, que deben generar la misma frecuencia, con una diferencia máxima del uno o el dos por ciento. A cada byte transmitido se le añade un "bit de comienzo", que se utiliza para que el reloj del otro extremo de la línea entre en sincronismo temporal con el reloj del emisor. Este sincronismo dura lo suficiente para permitir recibir

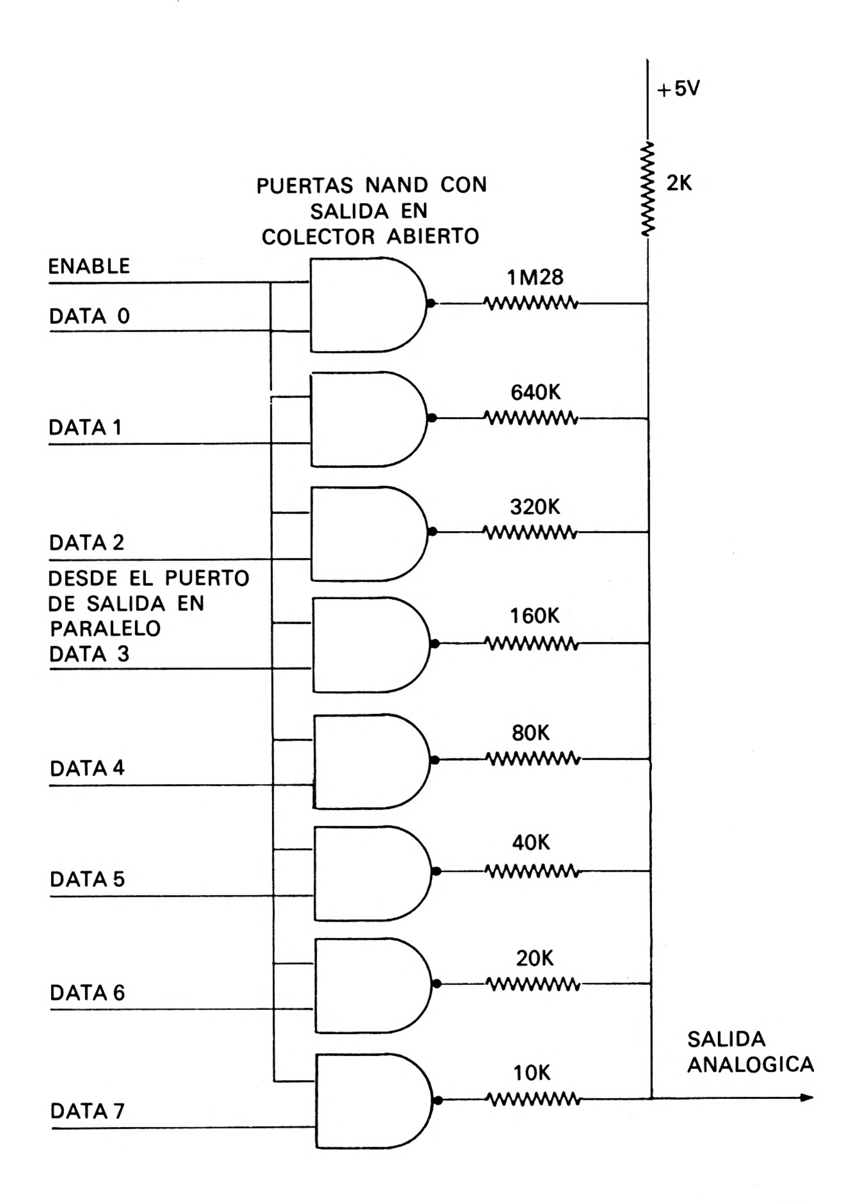


Figura 3.4. Convertidor analógico-digital simple.

correctamente los siguientes bits de datos. También se añade un "bit de parada", que se utiliza como tope entre la transmisión de un byte y el siguiente.

La frecuencia del reloj, en teoría, puede ser de hasta 9.000 Hz, quizá hasta 20.000 Hz en condiciones favorables, pero son valores más corrientes 110, 150, 300, 1.200, 2.400, 4.800, 9.600 y 19.200 Hz. (La velocidad de 110 Hz se utiliza con teleimpresoras.)

Los convertidores de los interfaces en serie tienen dos variedades principalmente: del tipo subordinado, que necesitan la ayuda de un ordenador para trabajar correctamente, y de tipo director, que realizan sus propias decisiones. Los de tipo subordinado se utilizan para salidas de ordenadores, pudiéndose poner en funcionamiento con varios formatos y velocidades, mientras que los del tipo director tienen estas opciones preestablecidas. Cuando no hay ordenador en una de las estaciones de comunicación debe utilizarse uno del tipo director.

El CPC464 no dispone de un interfaz en serie, pero se ha diseñado un interfaz externo. Este dispone de su propia ROM de control y, por tanto, no da problemas de *software* asociado. La creación de un sistema de este tipo puede entrañar muchos problemas, y es aconsejable no utilizar más de dos sistemas directores para un proyecto sencillo. Los detalles dependerán de los componentes elegidos.

Interfaz analógico

El método más simple de generar una salida analógica desde un ordenador se muestra en la figura 3.4. Esto se utiliza con éxito en la generación de sonidos, con datos almacenados en forma de onda. Realmente, la flexibilidad de estos sistemas es considerable, y la calidad del tono es muy superior a la obtenida utilizando ondas cuadradas.

Para esta aplicación no es necesaria demasiada exactitud en la relación entre la salida de datos y el voltaje producido. De hecho, una pequeña desviación puede mejorar la calidad del sonido. El método que se utiliza para generar los datos utiliza una tabla de valores preparada que representa los niveles instantáneos en un ciclo. La tabla puede establecerse mediante un corto programa en BASIC, pudiendo tener una onda sinusoidal o incluir armónicos.

Las distintas frecuencias se obtienen mediante una exploración de la tabla a diferentes velocidades. La velocidad puede permanecer constante, pero los incrementos del indicador varían. Si toda la tabla

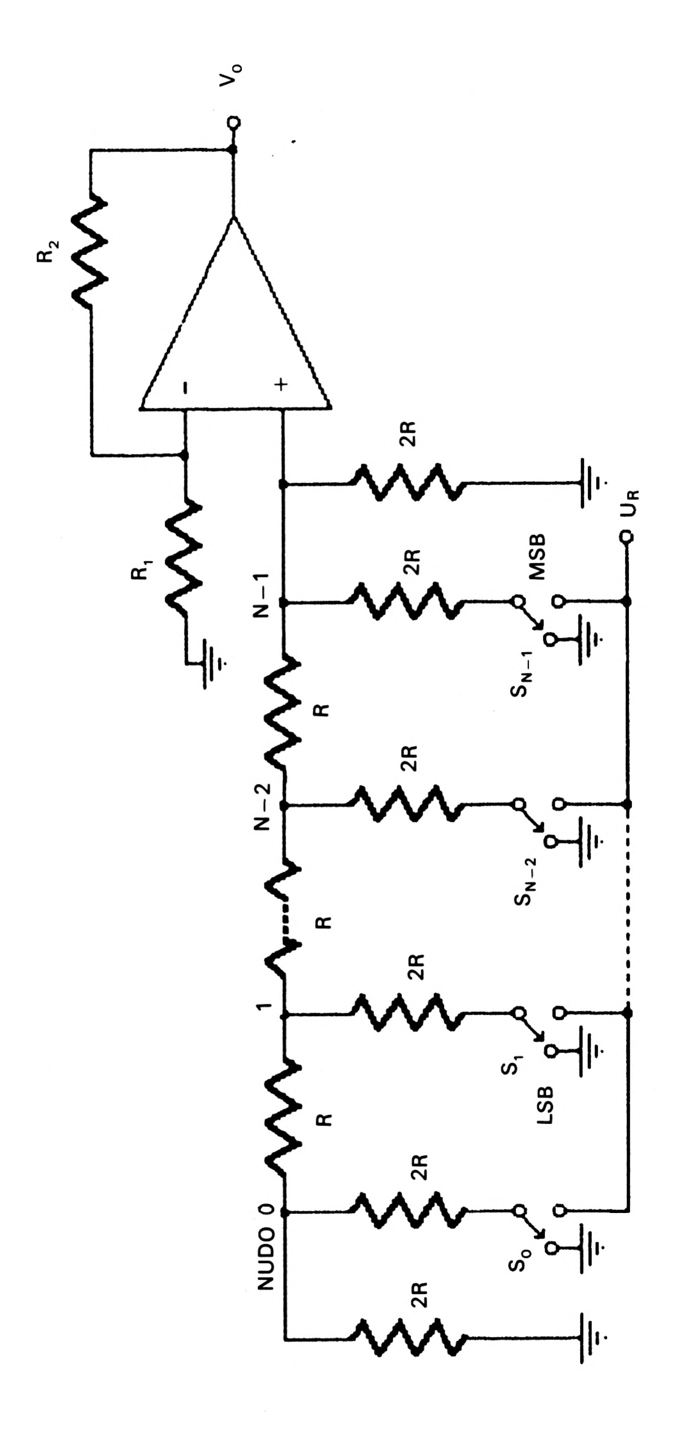


Figura 3.5. Convertidor D/A utilizando red de resistencias.

es explorada en 1/440 segundos, la nota producida es un A (la) por encima del C (do) medio.

Si se utilizan varios punteros y se suman los resultados, es posible generar una armonía. Sin embargo, como en todos los generadores de música, preparar los datos de control necesarios es muy pesado, aunque compensen los resultados.

El pionero de este sistema fue Howard Arrington, de Boise (Idaho), y sus muchas ramificaciones no pueden ser explicadas aquí.

Para una salida más precisa puede utilizarse el esquema de la figura 3.5, que utiliza un sistema escalonado y presenta la ventaja de que las resistencias sólo tienen los valores.

Una entrada analógica es un problema distinto. En teoría, es posible invertir una salida analógica controlando la parte digital desde un contador y parando cuando el voltaje generado sea igual al voltaje de entrada, pero es demasiado lento y complicado. En el extremo opuesto se pueden encontrar en el mercado componentes adecuados, con unas características y precio fabulosos. Entre medias hay también diferentes niveles de precios y características. Algunos dispositivos necesitan un soporte de *software*, especialmente aquellos que utilizan varios canales.

Un ejemplo simple para mostrar cómo opera un convertidor analógico-digital se lista a continuación:

```
100  INPUT D%
110  A%=128
120  B%=0
130  FOR X%+1 TO 8
140  B%=B%+A%
150  C%=D%-B%
160  IF C% < 0 THEN B%=B% XOR A%
170  PRINT A%;TAB(6);B%;TAB(12);C%
180  A%=A%/2
190  NEXT
200  PRINT B%
210  GOTO 100
```

El programa puede ser adaptado para realizar una entrada analógica de datos. Borra las líneas 100 y 160 y añade las líneas siguientes:

160 Manda B% a un puerto que controla un convertidor digital-analógico simple.

163 Introduce un bit desde un puerto; el bit depende del estado de un comparador controlado, por una parte, por la salida

del conversor D/A y, por otra, por el voltaje a examinar. Toma la entrada como C%, que es 0 si el voltaje externo es menor que la salida del convertidor D/A.

166 IF C%=0 THEN B%=B% XOR A%.

Se han omitido los detalles. Se ha facilitado todo el *hardware* y puedes elegir tus propias direcciones.

Este arreglo es más rápido que el acceso a un contador simple, que requiere 256 comparaciones en vez de 8. Es aconsejable comprobar que la salida del convertidor D/A sea razonablemente lineal; de otro modo pueden darse irregularidades. Para comprobar la linealidad, el método más imple consiste en mandar al convertidor los números de 0 a 225, haciendo una pausa en cada uno. Un polímetro conectado a la salida nos permitirá comprobar los incrementos.

Para obtener los mejores resultados de la conversión en ambas direcciones es muy importante utilizar dispositivos de precisión. Aunque experimentar mejorará los resultados, hay que hacerlo con cautela.

ROM auxiliares

Es casi inevitable que un sistema externo añadido al CPC464 requiera un soporte de *software*. El concepto general del sistema permite que este soporte se facilite con ROM montadas con el resto del *hardware* interno. Se puede añadir un máximo de 256 de estas ROM, pero en la práctica el método es el mismo para una ROM que para varias.

Primero, cada ROM tiene asignado un número. Una emisión de ese número por un canal de entrada/salida &CFXX debe habilitar la ROM. Esta reemplazará a la ROM superior interna en todo lo que concierne al sistema. Incluso lo hará la ROM 0 externa, aunque la interna se llama también ROM 0.

Una vez seleccionada la ROM externa pueden ejecutarse las rutinas que contiene. Es posible llamar o introducir rutinas de otras ROM externas, utilizando las ventajas que ofrece el "área RST", así como FAR CALL.

Ya que los programas almacenados en ROM externas pueden necesitar un área de trabajo, cada ROM deberá contener un programa de inicialización, que será llamado al conectar —o inicializar— el sistema por el sistema operativo principal. Este programa definirá la

cantidad de espacio en RAM que necesite, y los indicadores de los límites de RAM serán ajustados automáticamente.

Un programa almacenado en una ROM externa puede tener su propio juego de instrucciones, lo cual aumenta el número de palabras reservadas, que serán implementadas por una llamada a una rutina determinada.

Las ROM externas deben acomodarse a un formato específico, siendo algo complicada su utilización. Los detalles siguientes son lo más completos posible, pero algunas áreas permanecen oscuras debido a los datos disponibles. Sin embargo, se puede pensar que todos los puntos claves han sido tratados.

La mayor dificultad en la implementación de extensiones bien puede ser la creación de las ROM necesarias, aunque no es imposible utilizar RAM cargada desde el exterior y mantenida mediante baterías.

Tipos y formatos de ROM

Los programas en ROM pueden ser de tipo prioritario *(foreground)* o de tipo subordinado *(background)*. Los programas prioritarios tienen un control general, mientras que los subordinados se encargan de soportar funciones llamadas por el sistema prioritario utilizado. Por ejemplo, un programa subordinado puede proporcionar ventajas adicionales para controlar el sonido, en respuesta a una llamada del programa prioritario actual, pero puede que no sea capaz de ejecutar un programa completo por sí mismo. (En algunos casos, el sistema operativo puede verse como un programa subordinado, que realiza acciones en respuesta a las solicitudes del intérprete BASIC, pero a veces sólo actúa como un programa subordinado.)

Muchas extensiones periféricas pueden requerir un programa subordinado que las soporte. El programa puede ser tan simple como el descrito anteriormente para el puerto alternativo para la impresora, o tan complicado como un sistema operativo completo para disco.

Pueden utilizarse hasta siete ROM subordinadas, estando comprendidos sus números de referencia entre 1 y 7. En general, las ROM externas pueden tener los números de referencia entre 0 y 251. Los números de las ROM deben ser consecutivos, a partir de 0 o de 1. Si se utiliza la referencia 0, la ROM conectada estará disponible en el primer número de referencia no utilizado por las ROM externas.

Es posible usar hasta cuatro ROM con números de referencia consecutivos, para almacenar un programa prioritario simple.

Todo esto puede parecer un poco complicado, pero las dudas se irán disipando.

Una ROM externa puede ocupar las direcciones desde C000 hasta FFFF, pudiendo tener un tamaño máximo de 16K. Las seis primeras posiciones deben fijarse de este modo:

C000:	Tipo de ROM: 0 para una ROM prioritaria. 1 para una ROM subordinada. 2 para una extensión de ROM. (La ROM propia es del tipo &80.)
C001:	Número-marca de la ROM.
C002:	Número de la versión.
C003:	Nivel de modificación de la ROM.
C004/5:	Dirección de la tabla de instrucciones externas.

De C006 en adelante debe haber una tabla de saltos *(jumpblock)* comenzando por la rutina de inicialización y continuando con saltos que se ajustan a las instrucciones externas.

La tabla de instrucciones externas debe registrar las palabras de las instrucciones, sumando &80 al código de la última letra en cada palabra. Es aconsejable utilizar letras mayúsculas para dichas palabras de las instrucciones.

Una instrucción externa se reconoce porque lleva delante una barra vertical. Se implementa por KL FIND COMMAND en BCD4, al cual se entra con HL apuntando a la cadena de instrucciones, regresando con la dirección de selección de ROM y C y la dirección de la rutina requerida en HL. La tabla de instrucciones se termina con 0.

El uso de la tabla de instrucciones externas es opcional. El formato utilizado por el intérprete BASIC en ROM es:

C000 80 01 00 00 4C C0

Esto quiere decir que es la ROM propia: marca 0; versión 0; nivel de modo 0, y que la dirección de la tabla de instrucciones es C04C.

En C04C:

C04C 42 41 53 49 C3 00 : BASIC

El 0 final muestra que sólo hay una palabra de instrucción externa; por tanto, en lugar de iniciar el bloque de saltos en C006 es posible comenzar la rutina de inicialización.

Ahora pueden examinarse con más detalle algunas diferencias entre las ROM prioritarias y las subordinadas.

En el encendido inicial se inicializan los directores de los periféricos y del *firmware*, saltándose la ROM 0 en C006 con HL = ABFF (tope de la RAM disponible), DE = 0040 (base de la RAM disponible) y BC = B0FF (byte más alto utilizable). El puntero de la pila se inicializa a C000.

El programa de inicialización de la ROM 0 reserva espacio para incrementar en 012F el valor almacenado en DE, desplazando el límite inferior de la RAM disponible hasta 016F (los programas en BASIC se almacenan a partir de 0170). Los nuevos límites se almacenan en el área de trabajo.

Como ya sabrás, el tope de la memoria RAM libre puede modificarse desde BASIC mediante la instrucción MEMORY, facilitando un área para almacenar rutinas en RAM.

Si se llama a MC START PROGRAM con HL conteniendo 0000, se repite la inicialización como se indicó anteriormente; pero si la llamada se hace conteniendo HL otro valor, la ROM definida en C se introduce en la dirección apuntada por HL, con BC, DE y HL conteniendo los límites de memoria. El programa introducido necesitará reclamar algunas áreas de trabajo modificando adecuadamente DE y/o HL y almacenándolo para posibles referencias.

Observa que cuando se vuelve de la entrada a la ROM desde MC START PROGRAM se realiza una reinicialización completa del sistema; de este modo se vuelve a seleccionar la ROM propia como si se hubiera puesto en funcionamiento el ordenador.

El programa prioritario activado de esta manera puede elegir uno o más programas subordinados cuando los requiera como soporte. Esto puede hacerse llamando a KL ROM WALK, que a su vez llama a KM INIT BACK para todas las ROM subordinadas disponibles, o bien a INIT BACK para una ROM concreta requerida. La ROM del BASIC llama a KL ROM WALK, inicializando cualquier ROM que pueda estar disponible.

Los programas subordinados responden modificando los límites del área útil de RAM contenidos en DE y en HL, para reservar sus áreas de trabajo. Ahora bien, la localización de estas áreas dependerá de la cantidad de memoria reservada por el programa prioritario, y los programas subordinados deberán observar dónde comienza esta área y acceder a ella con un desplazamiento. Esto puede hacerse

utilizando IX como base. Si cada vez que se introduce el programa se prepara IX, es posible acceder a una posición concreta mediante LD A,(IX+n) o una instrucción similar.

Hasta ahora disponemos de un sistema que permite utilizar una o varias ROM prioritarias, cada una de las cuales puede contener varios programas a los que se accede mediante instrucciones externas, más un máximo de siete ROM subordinados, que pueden proporcionar rutinas de soporte. Una vez estemos en una ROM prioritaria debemos permanecer en ella, salvo para incursiones a las ROM subordinadas, ya que un retorno desde una ROM prioritaria origina una reinicialización general.

Aplicaciones

Los modos de utilización posibles de las ROM auxiliares son demasiado numerosos para examinarlos detalladamente, pero puede ser útil dar algunas indicaciones al respecto.

La provisión de lenguajes alternativos, como PASCAL o FORTH, es un punto de comienzo obvio, mientras otros programas podrían implementar procesadores de texto u hojas de cálculo. Hay una ventaja muy a tener en cuenta: que la RAM del ordenador utilizada se minimiza, requiriéndose sólo el área de trabajo.

Anteriormente se mencionó que la utilización de una RAM alimentada con baterías en vez de una ROM puede tener más alcance, ya que muchos datos se almacenan externamente, siendo trasladados a la RAM principal cuando sean requeridos. Presumiblemente no es necesario implementar técnicas de "memoria virtual", las cuales permiten ver una cantidad de RAM mucho mayor de la que en realidad se dispone. Los sitemas de ROM externa vienen a hacer una función similar.

Las ROM subordinadas pueden utilizarse para extender el sistema, ofreciendo un total de 112K de espacio de programa. Es probable, sin embargo, que parte de este espacio sea ocupado por un sistema de disco, un interfaz en serie u otros.

Es conveniente añadir una última observación: con un sistema tan complejo, pasará algún tiempo antes de explorarse completamente y descubrir las limitaciones escondidas. Se han detectado muchos pequeños "gazapos", en su mayoría triviales, como, por ejemplo, la inserción de un avance de línea cuando una línea de texto provoca una pérdida de información por extralimitación del ancho de panta-

lla. (Esta circunstancia puede ser deliberada, aunque no es corriente que así sea).

Ahora estamos más introducidos en el iceberg, y la imagen del CPC464 como una simple máquina de BASIC se nos antoja cada vez más remota. Puede utilizarse sólo de este modo, pero también puede hacer mucho más. El nivel de conocimientos que se necesita para utilizar todas las ventajas es bastante alto, y hay evidentemente mucho que estudiar.

"Hardware" de ROM externa

El *hardware* que se necesita para implementar una ROM externa no es demasiado complicado. Primero, las direcciones de salida de la forma DFXX deben ser reconocidas, y el byte de datos asociado debe ser transferido por un biestable *(latch)*. Este byte se compara con el número seleccionado de ROM para la ROM en cuestión, y si ambas comparaciones coinciden la ROM debe ser habilitada, pero sólo si la línea del interfaz $\overline{\text{ROMEN}}$ está en bajo. Todas las direcciones están disponibles para seleccionar una posición de ROM, y se esperará una transferencia de lectura con $\overline{\text{RD}}$ en bajo.

Las direcciones pertinentes pueden ser reconocidas con la ayuda de una puerta NAND de ocho entradas y de un inversor. El biestable puede ser uno estándar de ocho bits. La ROM puede ser casi de cualquier tipo. En la figura 3.6 se muestra una configuración posible, que utiliza comparadores para detectar el número seleccionado de ROM.

Si se utiliza RAM en vez de ROM, debe ser del tipo estático, para no necesitar refrescos.

No es necesario, por supuesto, montar las ROM en tarjetas separadas. Los elementos para reconocer la dirección y el biestable pueden ser comunes a más de una ROM, usando comparadores separados para detectar la ROM requerida.

Como en otras áreas de este libro, es una lástima no entrar en más detalles ya que las variaciones posibles son inmensas. Sólo con el circuito de la tarjeta podríamos escribir muchas páginas dando ideas.

Un segundo procesador

La técnica de utilizar el ordenador BBC como un "terminal inteligente" empleando un segundo procesador externo ha dirigido las investigaciones a la aplicación de una técnica similar al CPC464.

Haremos algunos comentarios, ya que sería muy complicado insertar una respuesta completa.

Primero, la necesidad de incluir un segundo procesador, es menor dadas las capacidades del CPC464, por su espacio extra de RAM y porque utiliza un procesador Z80, que es directamente compatible con CP/M. De todos modos, es posible enfocar situaciones en las que es útil llevar a cabo un proceso externo simultáneamente al que se está ejecutando en el CPC464. (El CPC464 puede cubrir la ejecución de dos procesos concurrentes alternando entre ellos, pero es imposible la ejecución simultánea de dos procesos al mismo tiempo.)

En un sistema lógico es importante distinguir entre un elemento director y un elemento subordinado. Sólo puede haber un director en un momento dado, aunque esta función puede ser transferida a otro elemento en cualquier instante. El director determina la acción del sistema, ejerciendo el control sobre los subordinados, los cuales pueden pedir una asistencia suscitando interrupciones.

Por definición, un procesador externo utilizado por el CPC464 como terminal inteligente debe ser el director, pero si ambos procesadores están trabajando independientemente, ambos pueden ser directores la mayor parte del tiempo, hasta que se requiera una comunicación entre ellos. Aquí es donde el procesador externo puede ejercer su autoridad, promoviendo una interrupción. Esto fuerza al procesador interno a responder, y la respuesta debe tener la forma de una transferencia de datos. Puede ser ejecutado por un programa subordinado, llamado mediante un suceso. El procesador externo puede pasar datos, definiendo la acción requerida.

La rutina que sirve las interrupciones del procesador externo ha de ser breve, ya que la acción normal interna debe mantenerse.

Un acercamiento alternativo, que puede tener sus ventajas cuando sólo es necesario pasar un dato, se basa en las líneas $\overline{\text{BUSRQ}}$ y $\overline{\text{BUSAK}}$. El procesador externo se apodera del *bus* interno durante un ciclo de memoria, y manda un byte de datos a la memoria interna. Esto puede repetirse para mandar un mensaje completo, pudiendo el procesador interno continuar con la ejecución entre las transferencias de datos, mientras el procesador externo prepara otro byte de datos.

El mensaje puede ser interpretado por un programa prioritario, pudiendo ordenar un cambio de acción del programa prioritario interno, etc. Esto puede implementarse en respuesta a un suceso síncrono llamado por el programa prioritario, que debe encargarse de que no se utilice un mensaje incompleto prematuramente.

Se ha expuesto lo suficiente para ver las posibilidades que ofrece

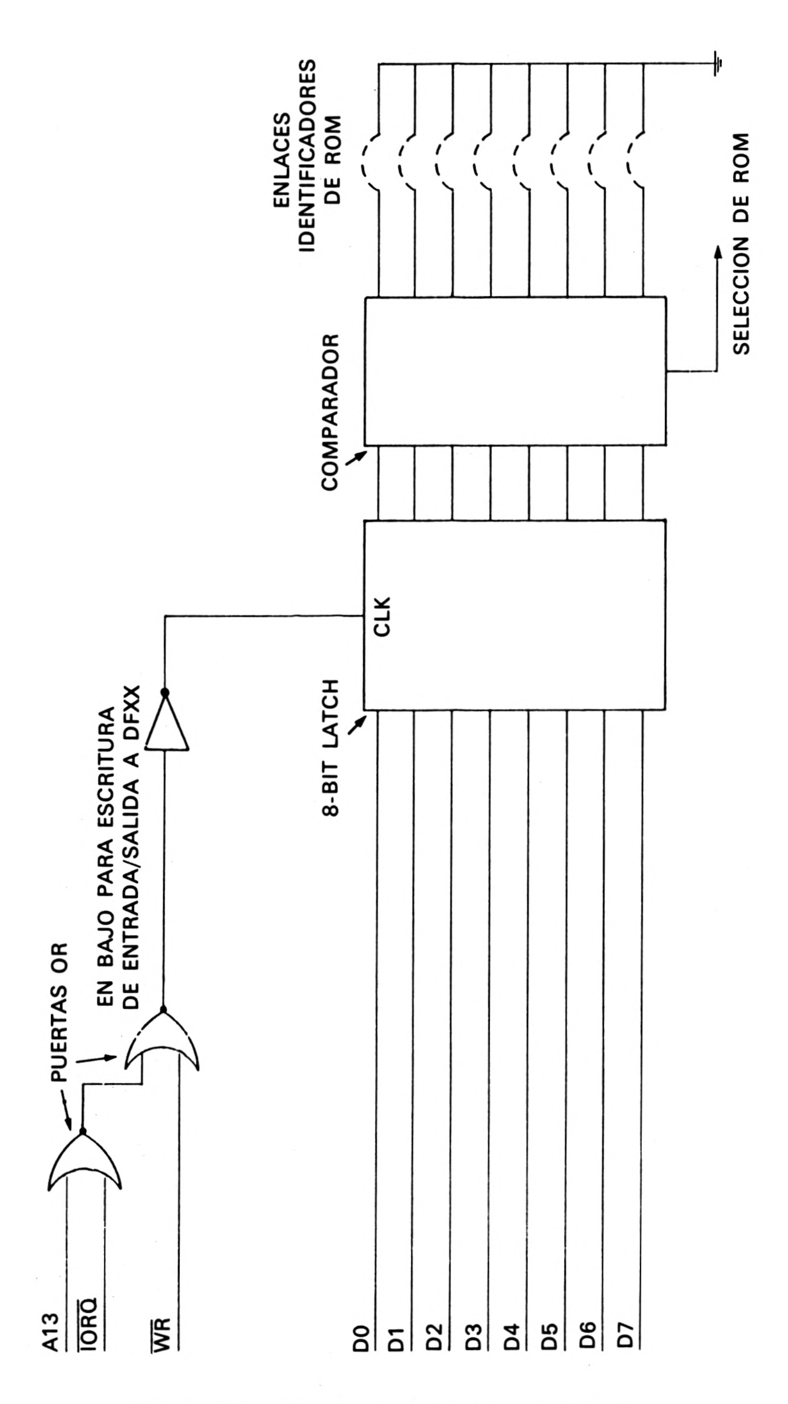


Figura 3.6. *Hardware* del selector de ROM.

el CPC464. Asimismo ha quedado evidenciado que sería necesario dedicar mucho más texto para enunciar con detalle los sistemas de *hardware* y de *software*. Un principiante podría encontrarse en el mismo caso de un conductor cuyo coche se ha averiado lejos de cualquier lugar habitado. Aunque posea las herramientas necesarias para efectuar la reparación carece de los conocimientos necesarios para utilizarlas correctamente.

El CPC464 facilita una elevada cantidad de "herramientas". Aprender a sacarles todo el partido posible llevará, sin duda, mucho tiempo.

Resumen

El CPC464 se ha revelado como un lobo disfrazado de oveja. Un observador superficial podría verlo como una máquina más de juegos, aunque ciertamente con bastantes características ventajosas. Una exploración más profunda revelará paulatinamente las múltiples ventajas que se esconden bajo su aspecto exterior.

Utilizar todo lo que hemos descubierto no es una tarea fácil. Trabajar con código de máquina es esencial, y una simple expansión externa requiere un cuidadoso diseño de *hardware*.

De todos modos, ésta es una máquina que servirá perfectamente a un usuario inquieto. Le ayudará mientras comienza a andar, y seguirá ayudándole cuando su caminar se convierta en carrera. A medida que se afiance en sus conocimientos, muy probablemente encontrará mejoras y periféricos para la máquina, lo cual le ayudará aún más a consolidar su aprendizaje.

Bibliografía

Programación del Z80, Rodnay Zacks (Anaya Multimedia).
Amstrad BASIC, AMSOFT.
Amstrad CPC464 Firmware, AMSOFT.
Amstrad DEVPAC for CPC464, AMSOFT.

Programa para la visualización o impresión de ROM y RAM

```
100  GOSUB 370
110  CLS
120  PRINT "MODO 0, Visualiza ROM
130  PRINT "MODO 1, Imprime ROM
140  PRINT "MODO 2. Visualiza RAM
150  PRINT "MODO 3, Imprime RAM
160  INPUT "MODO";Y
170  X=Y MOD 2
180  Z=Y/2
190  INPUT "Direccion de comienzo";A
200  A=A-65536*(A<0)
210  NL%=A-8*INT(A/8)
220  PRINT #(X*8), HEX$(A,4);
230  ON Z+1 GOSUB 310,460
240  PRINT #(X*8), TAB(6+3*NL%); HEX$(B,2);
250  A=A+1
260  NL%=NL%+1
270  IF NL%<8*(X+1) THEN 230
280  NL%=0
290  PRINT #(X*8)
300  GOTO 220
310  Q=INT(A/256)
320  POKE &7019,Q
330  POKE  &7018,(A-256*Q)
340  CALL &7000
350  B=PEEP(&7020)
360  RETURN
370  FOR X=&7000 TO &7012
380  READ Y
390  POKE X.Y
400  NEXT
410  RETURN
420  DATA &2A,&18,&70,&CD,&00,&B9
430  DATA &F5,&CD,&06,&B9,&7E
440  DATA &32,&20,&70,&F1,&CD,&0C
450  DATA &B9,&C9
460  B=PEEK(A):RETURN
```

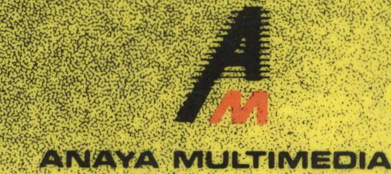
Indice alfabético

El AMSTRAD CPC464 es un ordenador único y muy especial. Una de las cualidades más importantes del AMSTRAD CPC464 es la facilidad de acceso a las rutinas más importantes de la ROM, ya sea usted un principiante o un programador profesional.

Don Thomasson analiza esta característica en los primeros capítulos de este libro de forma clara y bien estructurada, de forma que le permitirá escribir programas potentes y versátiles. Diseño de pantallas, entradas y salidas de cassette, etc., pueden ser reducidas a simples llamadas de subrutinas.

Otra de las cualidades más interesantes del AMSTRAD CPC464: su capacidad de comunicarse con el mundo exterior, y la posibilidad de ROM externa, que permiten aumentar la capacidad de su ordenador. Don Thomasson habla detalladamente de todo esto mostrándole, en suma, cómo aprovechar todas las posibilidades de su ordenador.

PROGRAMACIÓN AVANZADA DEL AMSTRAD, en definitiva, muestra la mejor forma de explorar esa parte del ordenador AMSTRAD que, de otra forma, permanecería escondida.



Programación Avanzada del AMSTRAD

Don Thomasson


AMSTRAD
CPC
OCR
MÉMOIRE ÉCRITE
MEMORY ENGRAVED
MEMORIA ESCRITA
ACME
https://acpc.me/